uc3m | Universidad Carlos III de Madrid

Ingeniería de Telecomunicaciones
2016/2017

Proyecto Fin de Carrera

Jupiter: Framework para el uso de modelos de Machine Learning

Miguel Fonseca Martínez

Tutor/es
Alejandro Baldominos Gómez
5 de Julio de 2017

A mi familia que me ha dado todo su apoyo.
A mi abuelo que le hubiese encantado poder verme convertido en ingeniero.
A Raquel, a la que tengo que agradecer todo el apoyo y empeño en que terminase y que ha tenido que soportarme hasta la saciedad.
A todos vosotros, gracias.

Índice

Índice de figuras

Índice de tablas

1. Introducción

El auge del **Deep Learning** en la sociedad actual, es un hecho.

En la era de la Información, vivimos rodeados de datos y el procesamiento de los mismos se convierte en una necesidad clave para poder realizar análisis detallado de las estrategias de negocio, y poder concretar resultados que puedan ser presentados a los accionistas o inversores.

Estos análisis siempre se basaban en datos pasados y con predicciones no exactas a largo plazo.

Sin embargo, desde hace unos años a ahora, y gracias a la mejora de la tecnología, y sobretodo, a la evolución de los dispositivos de computación, donde las GPU [1], han recogido el testigo de las CPU [2] en cuanto al procesamiento, ha permitido desarrollar algoritmos de aprendizaje basados en muchas capas o muchas neuronas (o capas convolucionales), llamado **Deep learning**.

Gracias a estos avances, se pueden entrenar sistemas que sean capaces de predecir datos, señales e imágenes, con una precisión increíble.

De hecho, es posible, gracias al Deep Learning, poder predecir potenciales enfermedades cardiacas, valores de bolsa futuros, reconocimiento de imágenes, generación automática de imágenes, y un infinito etc de posibilidades.

1.1. Objetivos

Aunque en la introducción de este proyecto, se ha hablado de Deep Learning, el objetivo de este proyecto no es el Deep Learning como tal, si no, proporcionar un Framework[3] y un API intuitivo[4]. A partir de un modelo desde cero, o partiendo de un modelo pre-entrenado, poder acceder de la manera mas sencilla al modelo y poder realizar estimaciones o predicciones del mismo, y gracias a este API accesible y sencillo de implementar, permitir al investigador y trabajador centrarse en mejorar su modelo, sin que le preocupe cómo implementarlo en un móvil o en la WEB.

1.2. Estructura del documento

Capítulo 1: Introducción

En este capítulo se detalla una pequeña introducción al tema del Proyecto, así como el objetivo del mismo. También recoge la estructura del mismo.

[1] GPU (Graphics Processor Unit) es un coprocesador dedicado al procesamiento de gráficos u operaciones de coma flotante, para aligerar la carga de trabajo del procesador central en aplicaciones como los videojuegos o aplicaciones 3D interactivas.

[2] CPU (Central Process Unit), 'unidad central de proceso', que es la parte de una computadora en la que se encuentran los elementos que sirven para procesar datos.

[3] Un framework, entorno de trabajo o marco de trabajo es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

[4] API (Application Programming Interface), es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Capítulo 2: Estado del Arte

En el Capítulo 2, se entrará en detalle sobre el estado actual de todas las tecnologías que se usan en el proyecto, y qué aspectos clave se puede destacar de cada una. Además, se realizarán comparativas con otras tecnologías igualmente similares, explicando los motivos por los que no se han empleado.

Capítulo 3: Requisitos

En el tercer capítulo, se expondrán una serie de requisitos funcionales, necesarios para la validación mínima del proyecto a la hora de su entrega o puesta en marcha.

Capítulo 4: Arquitectura

En este capítulo, se detallará la arquitectura del sistema a nivel global y cómo interactúan todas las tecnologías implicadas en el mismo.

Capítulo 5: Diseño

En el capítulo 5, se detallará el diseño del sistema, desde el modelo de base de datos hasta la elección del cliente, pasando por las decisiones implicadas en la interconexión de los diferentes elementos del proyecto propuesto. Se dividirá en dos grandes bloques, la parte automática, invisible para el usuario, y la parte en la que el desarrollador debe interactuar con el API.

Capítulo 6: Implementación

En este sexto capítulo, se tratará la implementación de los diferentes componentes del sistema, y se explicará el API propuesto, así cómo, las tecnologías implicadas y como integrarse con el mismo.

Capítulo 7: Validación y pruebas

En el capítulo 7, se explicará la validación del sistema y las pruebas a realizar para poder implementar el sistema en el entorno de Producción.

Capítulo 8: Planificación y Presupuesto

En este capítulo, se planificará el proyecto de varias etapas, y se presupuestará el mismo, en función del tiempo y recursos requeridos para poder crear una solución óptima.

Capítulo 9: Marco Regulador

En el noveno capítulo, se analizará el marco regulador que pueda afectar al proyecto.

Capítulo 10: Entorno socio-económico

En este décimo capítulo, se analizará el entorno socio-económico, para intentar entender la optimización de costes a realizar en el proyecto.

Capítulo 11: Conclusión

Por último, en este capítulo, se detallarán las conclusiones del proyecto y posibles futuras implementaciones del mismo.

2. Estado del Arte

En este capítulo, se van a analizar todas las tecnologías implicadas en el desarrollo del proyecto, así como un análisis de sus estado actual de desarrollo, escalabilidad y elegibilidad de las mismas. Se incluirán comparativas entre varias tecnologías y se explicarán aquellas que destaquen y se consideren importantes entre las analizadas.
Además, se presentarán conceptos necesarios para poder comprender de manera correcta, el contenido de este proyecto.

2.1. Deep Learning

Deep Learning[1] es un campo del *machine learning*[5] relacionado con algoritmos inspirados en la estructura cerebral, llamadas *Redes Neuronales Artificiales*.

Andrew Ng[6] describe el aprendizaje profundo dentro del contexto de las *Redes Neuronales* tradicionales, usando simulaciones como si se tratase de un cerebro real.

Sin embargo, la clave del Deep Learning ha sido la evolución tecnológica y la mejora superlativa en los procesos de computación, y gracias a ésto, se puede introducir suficiente cantidad de datos para entrenar grandes redes neuronales.

De hecho, otro de los puntos fuertes del Deep Learning, es que se pueden construir redes neuronales cada vez mayores y entrenarlas con más y más datos, y su rendimiento seguiría incrementándose.

Además de la escalabilidad, otro gran beneficio de los modelos Deep Learning es su habilidad para extraer de manera automática características a partir de los datos en bruto, a esta técnica se le llama, *feature learning*[7]

Es decir, los algoritmos *Deep learning* son capaces de estructurar una arquitectura desconocida basada en datos etiquetados, de manera que el algoritmo es capaz descubir otros rasgos asociados en términos de los ya conocidos.

Uno de los ambientes donde destaca el *Deep learning*, se encuentra en dominios donde las entradas o las salidas, son analógicas, es decir, aquellas que no son una serie de datos en cantidades pequeñas, sino que son imágenes (en píxeles), documentos (en texto) o audios.

Yann LeCun[8], es el padre de la arquitectura denominada *Convolutional Neural Network (CNN)*. Esta técnica se basa en perceptrones multicapa alimentados por redes neuronales. Además, la técnica escala con los datos y el tamaño del modelo y puede ser entrenado con *backpropagation*.

[5]Machine Learning es una disciplina científica del ámbito de la Inteligencia Artificial que crea sistemas que aprenden automáticamente

[6]Andrew Ng es un profesor asociado en el departamento de Ciencias de la Computación y del departamento de Ingeniería Electrónica por cortesía de la Universidad de Stanford, y trabaja como director del laboratorio de Inteligencia Artificial en Stanford. Es también el co-fundador de Coursera, la plataforma de educación en línea, junto con Daphne Koller.

[7]En Machine Learning, feature learning es un conjunto de técnicas que enseñan una característica, es decir, una transformación de datos de entrada a una representación que puede ser explotada en tareas de machine learning

[8]Yann LeCun (nacido en 1960) es un científico que destaca en las contribuciones en *machine learning*. Es bien conocido como creador de CNN, y como padre fundador de la misma.

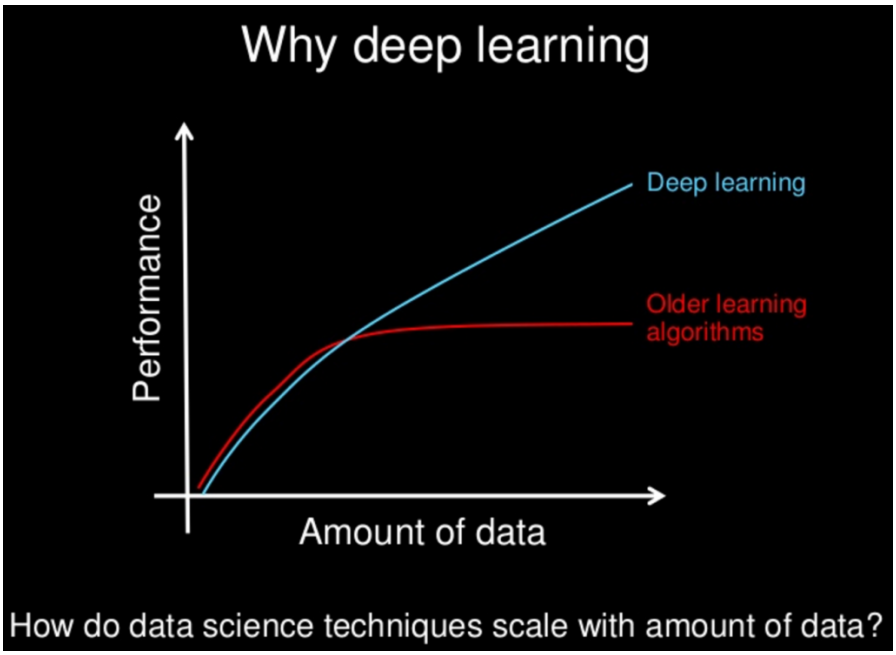

Figura 1: ¿Cómo escalan las técnicas de datos con la cantidad de datos?.[1]

De hecho, se pueden implementar grandes CNN's, que tienen un gran éxito en el reconocimiento de objetos o patrones en fotografías.

Otro de los padres del Deep Learning es Jurgen Schmidhuber[9], y es el creado de la arquitectura *Long Short-Term Memory Network (LSTM).*

Geoffrey Hinton[10] es otro de los personajes destacados en Deep Learning, ya que es uno de los primeros investigadores que demostraron el uso generalizado de las técnicas de *backpropagation*

Por último, otro personaje importante asociado con el Deep Learning, es Demis Hassabis [11], fundador de DeepMind, cuyo objetivo es combinar técnicas de Deep Learning con Reinforcement Learning, para manejar problemas complejos, como enseñar a una IA a jugar, quedando demostrados los pasos dados con éxito entrenando jugadores de los juegos Atari.

[9]Jürgen Schmidhuber es un científico que trabaja en el campo de la inteligencia artificial

[10]Nacido el 6 de Diciembre de 1947 is un científico, destacado por su trabajo en el campo de la inteligencia artificial

[11]Demis Hassabis (nacido el 27 de julio de 1976) es un investigador de inteligencia artificial, neurocientífico, diseñador de juegos de ordenador y jugador de clase mundial

2.1.1. Diferentes Frameworks para aplicar Deep Learning

Figura 2: Tensorflow

Tensorflow es un framework a bajo nivel para utilizar computación numérica usando grafos de datos. Fue desarrollado por Google's Machine Intelligence Research Organization
Soporta cuDNN[a] **v6** para aceleración GPU
Lenguajes soportados: C++, Python, Go, Java (Beta)

[a]The NVIDIA CUDA Deep Neural Network library (cuDNN) es una librería de funciones primitivas sobre GPU que provee implementaciones de rutinas estándar, como pueda ser convolución, pooling, normalización o capas de activación

Figura 3: Theano

Theano es un compilador de expresiones matemáticas que define, optimiza y evalúa expresiones matemáticas utilizando arrays multidimensionales.
Soporta cuDNN **v5.1 y v6** para aceleración GPU
Lenguajes soportados: Python

Figura 4: Keras

Keras es una librería minimalista y a alto nivel de Redes Neuronales.
Escrita en Python y capaz de ejecutarse encima de Tensorflow y Theano. Fue desarrollada para permitir la experimentación rápida.
La versión de cuDNN depende de la versión de TensorFlow o Theano instalado con Keras.
Lenguajes soportados: Python

Figura 5: Lasagne

Lasagne es una librería ligera y a alto nivel de Redes Neuronales.
Escrita en Python y se ejecuta sobre Theano, pudiendo además acceder a las variables a mas bajo nivel de Theano.
La versión de cuDNN depende de la versión Theano instalado con Lasagne.
Lenguajes soportados: Python

Figura 6: Caffe

Caffe es un framework de Deep Learning veloz, modular y basado en expresiones.
Está desarrollado por la BVLC (Berkeley Vision and Learning Center), asi como muchos contribuidores.
Caffe soporta cuDNN **v5** para aceleración GPU
Lenguajes soportados: C, C++, Python, MATLAB, Command line interface

Figura 7: Caffe 2

Caffe 2 es un framework de Deep Learning simple y flexible.
Programado sobre Caffe, está diseñado de manera más flexible, permitiendo organizar la computación de múltiples maneras.
Soporta cuDNN **v5.1** para aceleración GPU
Lenguajes soportados: C++, Python

Figura 8: Torch

Torch es un framework cientifico de computacion que ofrece soporte a algoritmos de aprendizaje máquina
Soporta cuDNN **v5** para aceleración GPU
Lenguajes soportados: C, C++, Lua

Figura 9: CNTK

Microsoft Cognitive Toolkit es un conjunto de herramientas diseñadas por Microsoft Research que permite, de manera mas sencilla, entrenar y combinar modelos a través de GPU y servidores.
Implementa de manera eficiente CNN y RNN.
Soporta cuDNN **v5.1** para aceleración GPU
Lenguajes soportados: Python, C++, C# y CLI

2.2. Cloud Computing

El *Cloud Computing*, comunmente llamado *La nube*, consiste en proporciona recursos de computación bajo demanda, que se ejecutan en *data centers*, sobre Internet y basado normalmente, en el concepto *Pay-per-use*, es decir, se paga por lo que se consume.

2.2.1. Infrastructure as a Service (IaaS)

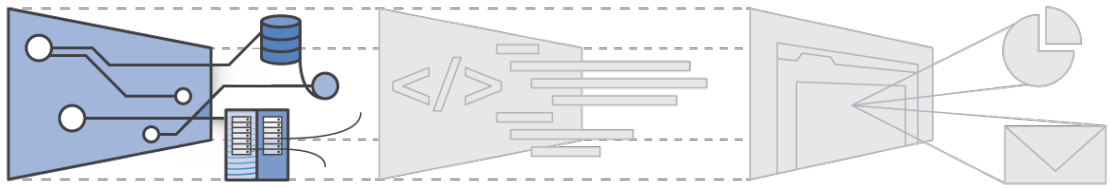

Figura 10: Infrastructure as a Service (IaaS).[4]

Definición

Insfraestructure as a Service (IaaS) es una infraestructura que proporciona bloques fundamentales para el desarrollo de proyectos. Ofrece acceso a las redes, a los equipos (ya sean máquinas virtuales, host dedicados, etc) y al almacenamiento de datos.

Proporciona un nivel de flexibilidad alto y de control, y puede llegar a ser familiar debido a la similitud con entornos ya conocidos.

Proveedores IaaS

Amazon AWS

Ofrece un gran espectro de ofertas de computación en la nube y almacenamiento, incluyendo instancias y servicios especializados como pueda ser Amazon Elastic Map Reduce o instancias con GPU's en cluster, asi como SSD's de alto rendimiento.

Ademas, la parte IaaS ofrece múltiples servicios, ya englobados como PaaS, como workflows, mensajería, auditoría, caché, servicios de búsqueda, bases de datos, etc.

Aspectos clave: Un abanico de servicios muy amplios y modelo de precios muy competitivo. Uso de una capa gratuita durante un año.

Limitaciones: AWS es complejo y si deseas integrar múltiples servicios, puede llegar a ser complicado estimar los costes.

Windows Azure

No es solo Windows como IaaS. Ofrece servicios de computación en Linux, Windows y otras plataformas, así como servicios de almacenamiento como cualquier otro proveedor.

Este IaaS ofrece acceso a redes virtuales, buses de servicios, colas de mensajería y bases de datos.

Aspectos clave: Herramienta de administración fácil de usar

Limitaciones: Al ser una herramienta visual de administración, es más complicado profundizar a más bajo nivel.

Google Compute Engine

Google Compute Engine esta diseñado para Big Data[12], almacenamiento de datos, computación de alto rendimiento y otras aplicaciones analíticas.

Está muy bien integrado con los servicios de Google. Además Google Compute Engine es relativamente nuevo, comparado con sus rivales, y la manera en la que se ejecuta en la infraestructura de Google, hacen de este servicio una herramienta muy potente.

Aspectos clave: Con la infraestructura de Google en mente, está diseñado para ser muy escalable.

Limitaciones: Falta de herramientas de administración.

2.2.2. Platform as a Service (PaaS)

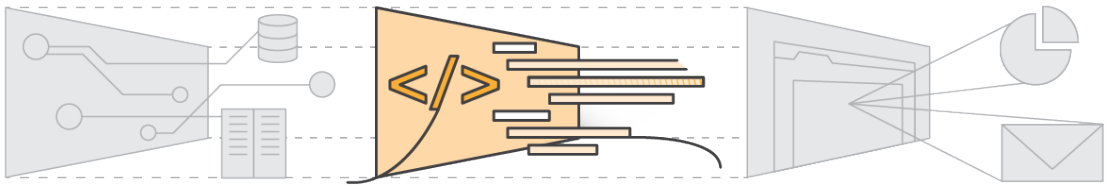

Figura 11: Platform as a Service (PaaS).[4]

Definición

Platform as a Service (PaaS) es una categoría de servicios de computación en la nube que ofrece una plataforma completa que permite desarrollar, ejecutar y administrar aplicaciones sin la complejidad de construir y mantener una infraestructura, íntimamente relacionada con el desarrollo e implementación de aplicaciones.

PaaS puede ser utilizado de dos maneras:

- Como un servicio público en la nube que ofrece un proveedor, donde el consumidor del mismo controla y mantiene el desarrollo de software basado en configuraciones mínimas, y donde el proveedor provee las redes, los servidores, el almacenamiento, el sistema operativo, el 'middleware', las bases de datos y otros servicios relacionados.
- Como un servicio privado dentro de un firewall (Red interna o aplicaciones internas)
- Como software desplegado en una infraestructura pública como servicio.

[12]Big data es el término utilizado para conjuntos de datos que son demasiado grandes o complejos como para procesarlos con el software tradicional

Ventajas y desventajas

Ventajas de PaaS

- **Desarrollo y testeo ágil**: Con PaaS, los equipos de desarrollo pueden probar y analizar diferentes configuraciones, con múltiples máquinas y diferentes localizaciones, de esta manera, se pueden ejecutar test de estrés y rendimiento, de compatibilidad y hallar respuestas en diferentes entornos, que serían imposibles de analizar en un entorno *local*.
  Es por eso, que todo el desarrollo se acelera y al optimizar el rendimiento del equipo de desarrollo, se ahorra.

- **Asignación dinámica**: Los equipos de desarrollo tienen que tener la flexibilidad de poder realizar test rápidamente y lanzar una nueva funcionalidad en una aplicación o servicio de manera ágil, con PaaS y la computación en la nube, estas tareas son más fácilmente implementables.

- **Centrarse en el negocio y el desarrollo**: Gracias a PaaS, las compañías o desarrolladores, ya no necesitan centrar esfuerzos en el mantenimiento y elección de sistemas, y por ello, se pueden centrar en el negocio.

Desventajas de PaaS

- **Seguridad de datos**: Como otras soluciones en la nube, muchas compañías tiene un nivel de confidencialidad más alto que el que es ofrecido por cualquier PaaS.
  Además, muchos negocios siguen siendo escépticos al no tener el control de sus aplicaciones, por ejemplo, las organizaciones gubernamentales necesitan tener asegurados los datos para cumplir las normativas de privacidad y seguridad que dictan las leyes del Estado en cuestión.

- **Flexibilidad limitada**: Las soluciones PaaS no tienen la misma flexibilidad que IaaS.
  Los usuarios de PaaS no pueden crear y borrar múltiples máquinas virtuales, y comparado con SaaS (Sofware as a Service), no ofrece un producto completo, por lo que la compañía tiene que poner esfuerzos en diseñar, crear y testear programas antes de ponerlos en producción.

- **Dependencia**: Con un número limitado de proveedores PaaS, aquellos que quieran usarlos tienen que aceptar sus condiciones

- **Problemas de integración con sistemas propios**: La integración de PaaS con el resto de sistemas propios puede ser compleja.

Proveedores PaaS

Red Hat Openshift

Red Hat OpenShift esta basado en aplicaciones OpenSource y ofrece una gran variedad de lenguajes, bases de datos y componentes.
Este PaaS es altamente personalizable y se ofrece de tres maneras:

- OpenShift Online: hosting online
- OpenShift Enterprise: Un PaaS privado que se ejecuta en un centro de datos propio
- OpenShift Origin: La plataforma OpenSource de RedHat Openshift

Openshift automatiza la administración de tareas, permite escalabilidad y soporta repositorios.

Aspectos clave: Alto numero de opciones para el conjunto de aplicaciones, desde el backend hasta el frontend. Los desarrollares pueden interactuar con OpenShift a través de una consola web, la línea de comandos o con IDE.

Limitaciones: Se integra bien con Git, pero con los demás repositorios no funciona demasiado bien.

Google App Engine

Google App Engine esta diseñado para la aplicaciones web distribuidas desarrolladas en Java, Python, PHP y Go.

En entorno Java soporta otros lenguajes que hagan uso del JRE y hay un SDK para uno de los lenguajes, así como un plugin para Eclipse.

Este PaaS ofrece una infraestructura que garantiza que es escalable, aunque solo si se cumplen las restricciones impuestas.

Además, tiene bases de datos, esquemas basados en clave-valor, que magnifican la capacidad de escala de la infraestructura.

Aspectos clave: Google App Engine usa un modelo sandbox, reduciendo el riesgo de procesos extraños ejecutados en el servidor físico que destruya o entorpezca otros procesos en el servidor.

Limitaciones: Los lenguajes de programación están limitados a Java, Python, Go y PHP

Heroku

Heroku soporta Ruby, Python, Java, Scala, Cloture y Node.js.

La plataforma provee un entorno abstracto de computación, denominado “dynos”, que se trata de contenedores Unix que ejecutan procesos en entornos aislados.

Hay múltiples tipos de Dynos, como Web Dynos (atiende peticiones HTTP) o Worker Dynos (atiende peticiones de tareas en una cola). También tiene *addons*, que se pueden usar como servicios integrados.

Aspectos clave: Es uno de los primeros proveedores PaaS, es ideal para despliegues rápidos.

Limitaciones: Estimar los costes puede ser complicado.

Amazon AWS

Aunque AWS es principalmente un IaaS, muchos de los servicios disponibles pueden considerarse PaaS, ya que puedes usar los servicios disponibles sin tener que preocuparte de crear o mantener tus propios servidores de aplicaciones.

Soporta Java, Python, Ruby y Perl, entre otros. Además, ofrece RDS, que es una base de datos sin necesidad de administración.

Por último, gracias a Amazon Elastic Beanstalk se pueden desplegar aplicaciones auto-escaladas, auto-balanceadas y monitorizadas.

Aspectos clave: Al tratarse de una extensión de IaaS, no hay limitación de lenguajes, bases de datos o tecnologías de servidor que puedes instalar y ejecutar.

Limitaciones: Requiere mas administración que otros PaaS

2.2.3. Software As A Service (SaaS)

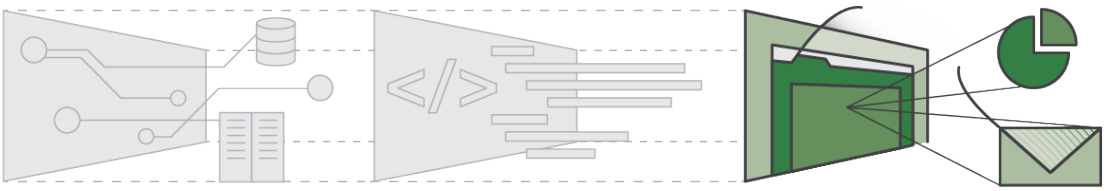

Figura 12: Software As A Service (SaaS).[4]

Definición

El software como servicio proporciona un producto completo. Normalmente, quien habla de SaaS, se refiere, en realidad, a aplicaciones finales de usuario.

Gracias a SaaS, el usuario no tiene que preocuparse en cómo se mantiene el servicio y la infraestructura, solamente tiene que preocuparse de cómo utilizar el software concreto.

Sin embargo, tiene la desventaja que el servicio lo presta el fabricante, por lo tanto, la compañía o desarrollador no puede desplegar software propio.

2.3. Servidores

2.3.1. Arquitectura de cliente-servidor

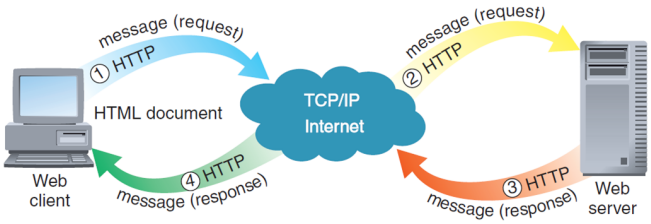

Figura 13: Arquitectura Cliente - Servidor.[10]

Definición

El modelo cliente-servidor es una arquitectura para aplicaciones distribuidas, y se divide entre proveedores de recursos o servicios, llamados servidores, y peticionarios de servicios, llamados clientes.

Habitualmente los clientes y los servidores se comunican a través de una red en Hardware separado, pero ambos, cliente y servidor, pueden estar en el mismo sistema (en entornos de desarrollo, por ejemplo).

Un servidor puede ejecutar una o más aplicaciones, que comparte sus recursos con los clientes.

Un cliente no comparte sus recursos, pero sí realiza peticiones de contenido a los servidores, además inician sesiones con los servidores, los cuales esperan las peticiones.

Rol cliente-servidor

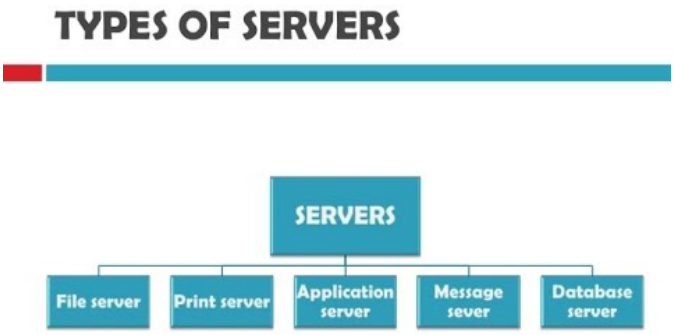

Figura 14: Tipos de Servidores.[10]

La arquitectura basada en el paradigma cliente-servidor describe la relación entre programas en una aplicación. El componente servidor provee un servicio a uno o varios clientes, que solicitan dichos servicios. Los servidores están clasificados según el servicio que proveen, por ejemplo, un servidor web provee páginas web, o un servidor de archivos, provee archivos.

Si una máquina es un cliente, un servidor o ambos, es determinado por la naturaleza de la aplicación que requiere el servicio.

Por ejemplo, una máquina puede ejecutar un servidor web y un servidor FTP al mismo tiempo, y atender diferentes peticiones según la demanda. Además, el cliente puede comunicarse con un servidor en la misma máquina. Un caso peculiar es la comunicación entre servidores, para sincronizar sesiones o datos, llamada comunicación "server-to-server"

Comunicación cliente-servidor

En general, un servicio es una abstracción de los recursos de un sistema, y el cliente no debe saber cómo el servidor funciona, y el rendimiento del mismo cuando realiza una petición.

El cliente solo tiene que entender la respuesta y que los protocolos concuerden.

Los clientes y servidores intercambian mensajes con un patrón *request-response (Ver figura 15)*.

El cliente envía una petición, y el servidor devuelve una respuesta, ajustándose a los protocolos definidos en la comunicación.

Todos los protocolos cliente-servidor operan en la capa de aplicación.

Para formalizar el intercambio de datos, el servidor puede implementar un API. Un servidor puede recibir peticiones desde diferentes clientes en un corto periodo de tiempo, esto implica que el servidor puede llegar a colapsar.

Para ello, el servidor puede limitar la cantidad o disponibilidad de los clientes.

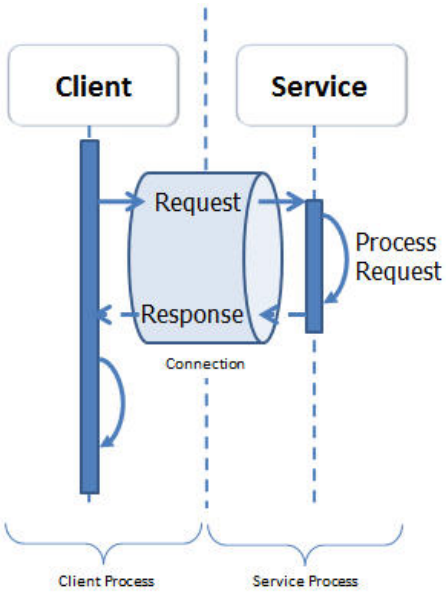

Figura 15: Patrón *Request-Response*.[11]

2.3.2. Diferentes modelos de arquitectura cliente-servidor

Model View Controller (MVC)

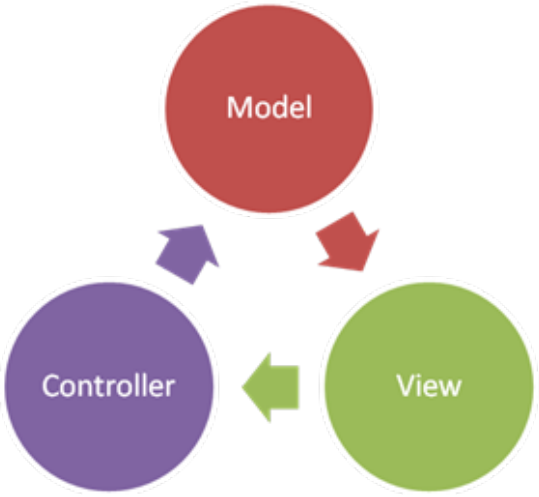

Figura 16: Model View Controller

En la programación orientada a objetos, el modelo MVC es el nombre de un patrón de diseño para relacionar de manera eficiente la interfaz de usuario con los modelos de datos subyacentes.

Está muy expandido en el desarrollo de aplicaciones WEB y programas, en general.

El patrón MVC ha sido señalado por muchos desarrolladores como una gran utilidad para reutilizar objetos y código, que permite la reducción del tiempo que conlleva desarrollar aplicaciones con interfaces de usuario.

MVC propone tres componentes principales:

1. **El modelo (M)**: Representa la estructura lógica de los datos en software, y no contiene información sobre la interfaz de usuario
2. **La vista (V)**: Es una colección de elementos que estructuran la interfaz de usuario
3. **El controlador (C)**: Representa los objetos y elementos que interconectan la vista y el modelo

Model View ViewModel (MVVM)

El patrón Model-View-ViewModel puede ser usado en todas las plataformas. Su objetivo es ofrecer una separación clara entre la interfaz de usuario y su lógica.

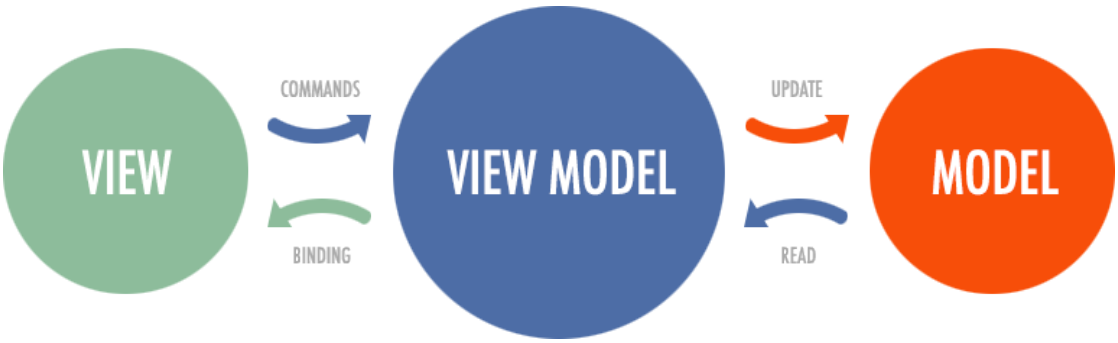

Figura 17: Model View ViewModel.[12]

Hay tres componentes en el patrón MVVM:

1. **El modelo (Model)**
2. **La vista (View)**
3. **El modelo de vista (ViewModel)**

Estos tres componentes están desacoplados entre ellos, permitiendo:

- Los componentes puede ser intercambiados
- Una implementación interna en cada componente, puede ser hecha sin afectar a los otros
- Los componentes pueden trabajar independientemente.
- Testeo unitario por componente aislado (sin dependencias de los demás)

Además de entender la responsabilidad de cada uno de los tres componentes, también es importante entender cómo los componentes interactúan entre ellos. A más alto nivel, el componente *view* conoce la vista, el *view-model*, conoce el modelo de la vista, sin embargo, vista y modelo son independientes uno del otro.

View

La vista es la responsable de definir la estructura, *el layout* y la apariencia de aquello que ve el usuario en la pantalla, con un código limitado que no contiene lógica de negocio.

Model

El modelo es una implementación del modelo que incluye el modelado de datos a través de una lógica de negocio y lógica de validación. Por ejemplo, los modelos incluyen repositorios, objetos de negocio, *data transfer objects (DTO's)* y entidades generadas u objetos proxy.

View Model

El view model actúa como una capa intermedia entre la vista y el model, y su responsabilidad es tener la lógica de vista.

Típicamente, la vista interactúa con el modelo invocando métodos en las clases del modelo.

El view model provee datos del modelo de manera que sea más simple su utilización en la vista. Provee implementaciones de comandos que el usuario de la aplicación inicializa en la vista.

Además, utiliza el concepto de *two-way data binding* con la vista.

2.4. Clientes

2.4.1. Clientes WEB

El cliente, o el lado usuario de la Web. Normalmente se refiere al navegador web. También se puede referir a plugins o aplicaciones auxiliares que mejoran el navegador, permitiendo soporte especial a servicios desde la Web.

También puede implicar la máquina completa de un usuario, o al dispositivo que accede a la web.

Angular 4

Angular 4 es un framework de front-end basado en componentes. Un componente es la combinación de una plantilla HTML y un controlador de dicho componente, que se encarga de manejar los eventos, atributos y funciones del mismo.

- Multiplataforma: Con Angular 4 se pueden desarrollar aplicaciones multi-plataforma, ya que al ser un framework basado en componentes, se pueden reutilizar para cualquier plataforma.
- Rápido y eficiente: Esta cuarta versión del framework se centra en mejorar la carga de los componentes, y en disminuir el consumo de recursos.
- Desarrollo ágil: El uso de componentes permite la reusabilidad y por tanto, agiliza los desarrollos.
- Comunidad amplia: Una gran comunidad que permite que evolucione el framework de manera estable.

2.4.2. Clientes de Escritorio

Cliente que se ejecuta en una máquina y todo el peso de la aplicación se encuentra instalado localmente.

2.4.3. Clientes móviles

Análogo al cliente de Escritorio, pero utilizando una plataforma móvil

2.5. Protocolos WEB

El protocolo estándar de Internet es TCP/IP [13]. La parte IP del protocolo se refiere a la manera de enrutar o direccionar los paquetes de datos. Sobre esta capa TCP/IP actúan los protocolos HTTP y FTP. Cada uno de ellos tienen diferentes funciones aunque gracias a su cooperación se pudo desarrollar la *World Wide Web*

2.5.1. Protocolo HTTP

El protocolo HTTP [14] es un protocolo situado en la capa de aplicación para la distribución de contenido. Es la base de la comunicación de la *World Wide Web*

El *Hypertext* es texto estructurado que usa enlaces lógicos entre los nodos que contienen texto. HTTP es el protocolo para el intercambio de estos enlaces

Funciona como un protocolo de petición-respuesta en un modelo cliente-servidor. Por ejemplo, un navegador web puede ser el cliente y una aplicación ejecutándose en un servidor puede ser el servidor. El cliente solicita contenido al servidor, y éste responde con un recurso como pueda ser un archivo HTML y otro contenido.

Un navegador web es un ejemplo de un agente de usuario (UA).

2.5.2. Protocolo HTTPS

El protocolo HTTPS [15] es un protocolo de comunicación utilizado para comunicaciones seguras sobre una red, usado sobretodo, en Internet. Consiste en una comunicación HTTP en la cual, la conexión está encriptada por una capa de transporte segura. La principal motivación para HTTPS es la autenticación y protección de la integridad de los datos intercambiados.

2.6. Rest API

2.6.1. Definición

RESTful API es una interfaz de aplicación que usa peticiones HTTP para obtener y almacenar datos, mediante los métodos GET, PUT, POST y DELETE.

[13] Transmission Control Protocol/Internet Protocol
[14] Hypertext Transfer Protocol
[15] HTTP over Transport Layer Security (TLS)

2.6.2. Características

La tecnología REST es el equivalente a SOAP [16], sin embargo, es preferida debido a que utiliza menor ancho de banda, siendo preferible para un uso en Internet.

El API RESTful utiliza la filosofía *divide y vencerás*, en la cual divide una transacción en módulos más pequeños, lo que permite a los desarrolladores trabajar con mayor flexibilidad. Además, obtiene ventaja de las metodologías HTTP:

- GET: Para obtener un recurso
- POST: Para crear un recurso
- PUT: Para actualizar un recurso
- DELETE: Para eliminar un recurso

Por último, al tratarse de una metodología *stateless* es muy útil a la hora de ser escalados y redesplegados en caso de errores.

2.7. Frameworks JAVA

2.7.1. Definición

Los Frameworks son grandes estructuras de código escrito que se puede añadir al código propio para solucionar un problema en un dominio específico. Se puede usar un framework llamando a sus métodos, utilizado su herencia y proporcionando *callbacks* o cualquier otro patrón.

La diferencia entre Framework y Librería es que un framework utiliza IoC [17], es decir, extendiendo o invocando diversos métodos, es el Framework el que realiza todo el trabajo, mientras que una Librería proporciona unos métodos para facilitar diversas tareas. Sin embargo, en la actualidad, se utiliza la denominación Framework a aquello que congrega múltiples librerías y es amplio.

2.7.2. Spring

Spring es un framework de aplicaciones e IoC desarrollado para Java. Las características de este framework pueden ser usadas por cualquier aplicación Java, sin embargo, el principal objetivo del framework es su utilización para el diseño e implementación de aplicaciones web sobre Java EE. Además este framework no impone ningún modelo de programación, por lo que es muy popular como alternativa a EJB [18] y es *open source*

[16] Simple Object Access Protocol
[17] Inversion of Control
[18] Enterprise Java Beans

2.7.3. gRPC

gRPC es un framework RPC [19] *open source* de alto rendimiento que puede ser ejecutado en cualquier entorno. Conecta de manera eficiente servicios que se encuentren en múltiples servidores son capacidad para balanceo, trazabilidad y autenticación.

Los principales escenarios de uso son:

- Conectar de manera eficiente microservicios escritos en diferentes lenguajes
- Conectar dispositivos móviles y navegadores con servicios backend

Además tiene clientes en 10 lenguajes de programación, altamente eficiente y utiliza HTTP versión 2

2.8. Maven

Apache Maven es una herramienta de gestión de software y librerías en proyectos. Basado en el concepto POM [20], Maven gestiona la construcción de un proyecto, las librerías, las dependencias y la documentación.

2.9. Colas de Mensajería

2.9.1. Definición

La mensajería es una forma de comunicarse entre aplicaciones, y puede enviar mensajes y recibir mensajes de otro cliente.

Para ello, ambos se han de conectar a unos agentes de mensajería y escucha si tiene mensaje, o enviarlos. A efectos de comprensión del lector, una cola de mensajería es un tubo en el cual se dejan mensajes y el destinatario, que está al otro lado del tubo, los procesa.

La principal ventaja, es que ni el emisor ni el receptor tienen que estar conectados simultaneamente para comunicarse, de hecho, no tienen por qué saber nada uno del otro. Simplemente deben saber el formato del mensaje y el destino del mismo.

La mensajería se utilizará en las siguientes circunstancias:

- Componentes que no dependan de la interacción con otros componentes
- Que la aplicación funcione independientemente de si todos los componentes se ejecutan de manera simultanea
- El modelo permite a un componente enviar información de manera asíncrona.

[19] Remote Procedure Call
[20] Project Object Model

2.9.2. Tipos de colas

Point to Point (PTP)

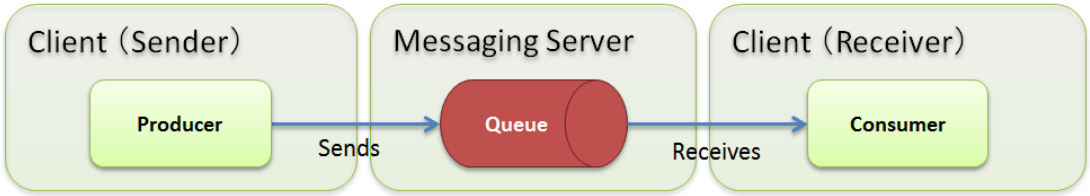

Figura 18: Mensajería PTP

En el tipo de cola PTP (Point to Point), un mensaje es consumido por un único receptor, pero puede haber múltiples emisores.

Se trata de un modelo FIFO, en el que el mensaje encolado será el primero en desencolarse.

Para permitir un manejo más correcto de la mensajería PTP, se ha añadido un nivel de prioridad, permitiendo priorizar aquellos con mayor nivel, y luego procesar los siguientes (FIFO con prioridad).

Una vez el receptor extrae el mensaje, envía un ACK para confirmar su correcta recepción.

Una aplicación de este tipo, se construye con el concepto de Productor/Consumidor. El emisor produce mensajes, y el receptor los consume, hasta que la cola se queda vacía o caduquen.

Subscription (Pub/Sub)

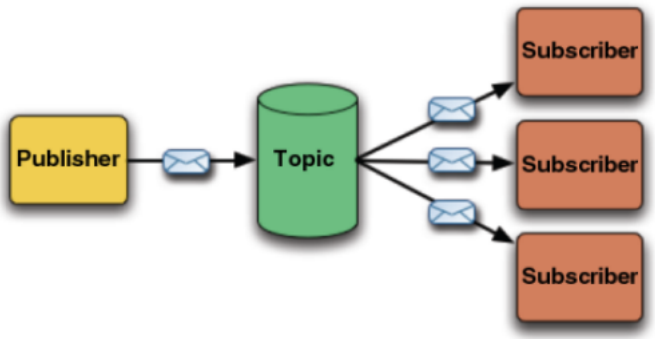

Figura 19: Mensajería Pub/Sub

En el modelo Pub/Sub, un mensaje puede ser consumido por múltiples consumidores. El destino del mismo es un tópico, no se encolan como PTP, sino, que aquellos subscriptores que estén suscritos al tópico, recibirán el mensaje y lo procesarán como buenamente puedan.

Normalmente, publicadores y subscriptores son anónimos y pueden, de forma dinámica, suscribirse o publicar contenidos. Para informar a los subscriptores se utiliza un mecanismo push(cita)

La mensajeria Pub/Sub tiene las siguientes características:

- Un mensaje - Múltiples consumidores (1 a N)
- Existe dependencia temporal entre Publicador y Subscriptor. El subscriptor tiene que estar conectado para poder recibir publicaciones, en el modo más estricto (*non-durable*)
- El API JMS ha permitido disminuir esta dependencia temporal, permitiendo crear suscripciones duraderas (*durable*), y permite recibir mensajes que fueron enviados mientras un subscriptor no estaba conectado.

Este modelo es especialmente útil cuando un grupo de aplicaciones quiere notificar a otras aplicaciones

2.9.3. Proveedores

IBM MQ

IBM MQ es un sistema de mensajería que agiliza la integración de diversas aplicaciones. Utiliza colas de mensajes para ofrecer una solución de mensajería para múltiples entornos.

Esta disponible como software distribuido autónomo, tanto en dispositivo físico como en IMB z/OS. Además, las cargas de trabajo se pueden enviar al cloud para su procesamiento.

ActiveMQ

ActiveMQ es la versión de Apache de colas de mensajería. Entre las características destacables, se encuentra:

- Es Open Source
- Es una implementacion de JMS, pero también proporciona API's para PHP, C/C++, .Net y Ruby
- Soporta HTTP y TCP
- Tiene una interfaz gráfica de administración

TIBCO EMS

Tibco EMS es el sistema de mensajería de Tibco. Soporta tanto colas como tópicos. Es caro, por lo que es recomendable para escenarios empresariales.

Amazon SQS

Amazon Simple Queue Service (SQS) es el servicio de colas de mensajes de Amazon.

Es un servicio completamente administrado para conseguir lograr una comunicación fiable entre componentes de software distribuido.

Con SQS, se pueden enviar, almacenar y recibir mensajes entre componentes, sin perderlos y sin acceder a otros servicios para lograr disponibilidad permanente.

Además, proporciona un SDK y un API de CLI

2.10. Contenedores

Figura 20: Docker.[25]

2.10.1. Definición

Docker es el contenedor de software líder en todo el mundo.

Los desarrolladores utilizan Docker para eliminar los problemas de desarrollo local cuando están implementando un proyecto.

Los operadores usan Docker para ejecutar y mantener aplicaciones en contenedores aislados, y así permitir una mejor optimización de recursos.

Las empresas usan Docker para implementar puestas en producción ágiles mediante tuberías, o para introducir nuevas características en un software de manera mucho mas rápida, segura y confidente.

Al usar contenedores, todo lo necesario para realizar un software está empaquetado en estos. Al contrario que las máquinas virtuales, los contenedores no son un sistema operativo completo, solamente contienen las librerías y configuraciones necesarias para que un software funcione. Esta característica hace que los contenedores sean eficientes, ligeros, sistemas auto-contenidos y garantiza que el software siempre será el mismo, sin importar donde esté desplegado

2.11. Integración Continua

2.11.1. Definición

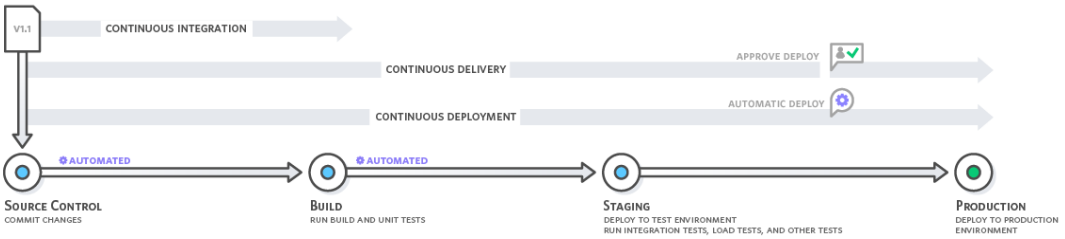

Figura 21: Integración Continua.[26]

La integración continua es un tipo de metodología en el desarrollo de software en la cual, los desarrolladores combinan cambios en el código en un repositorio central, donde se ejecutan versiones y pruebas automáticas.

Esta metodología se refiere a la fase de creación de un proyecto, y conlleva un componente de automatización y un componente cultural. Los objetivos de la integración consisten en encontrar errores y arreglarlos de manera ágil, así como mejorar la calidad del software y reducir el tiempo de validación, así como nuevas actualizaciones de software

Con la integración continua, los desarrolladores envían los cambios de forma periódica a un repositorio compartido con un sistema de control de versiones. El servicio de integración continua detecta los envíos al repositorio compartido y crea y ejecuta de forma automática pruebas de unidad en los cambios en el código para detectar al instante cualquier error funcional o de integración.

3. Requisitos

3.1. Requisitos funcionales

Se presentan los requisitos funcionales de la aplicación, según prioridad y fase de diseño. Los requisitos de la Fase 1 se basan en la funcionalidad mínima que ha de cumplir la aplicación para que sea utilizable como plataforma para el uso de modelos pre-entrenados. Los requisitos de la Fase 2 cubren la necesidad de poder entrenar modelos Inception V3 como base, siendo estos generalizables a otros modelos en las últimas fases del desarrollo.

3.1.1. Fase 1

Identificador	Descripción	Prioridad
RF1ID1	Creación de Roles, Grupos y Usuarios predefinidos	Alta
RF1ID2	El usuario Administrador, perteneciente al grupo ADMIN, puede crear roles	Alta
RF1ID3	El usuario Administrador, perteneciente al grupo ADMIN, puede crear grupos	Alta
RF1ID4	El usuario Administrador, perteneciente al grupo ADMIN, puede dar de alta usuarios	Alta
RF1ID5	El usuario Administrador, perteneciente al grupo ADMIN, puede asignar usuarios a grupos	Media
RF1ID6	El usuario Administrador, perteneciente al grupo ADMIN, puede asignar roles a grupos	Media
RF1ID7	Un usuario puede darse de alta y se le asigna el grupo JUNO	Alta
RF1ID8	Creación de modelos predefinidos y habilitar los mismos	Alta
RF1ID9	Usuarios con permisos pueden crear un modelo pre-definido usando su propio modelo pre-entrenado	Alta
RF1ID10	Usuarios con permisos pueden arrancar sus modelos pre-definidos	Alta
RF1ID11	Usuarios con permisos pueden parar sus modelos pre-definidos	Alta
RF1ID12	Usuarios con permisos pueden probar sus modelos pre-definidos	Alta
RF1ID13	Usuarios con permisos pueden eliminar sus modelos pre-definidos	Alta
RF1ID14	Desarrollador puede conectarse con su modelo mediante API Java	Alta
RF1ID15	Desarrollador puede conectarse con su modelo mediante API Swift	Alta

Tabla 1: Requisitos Funcionales de la Fase 1

3.1.2. Fase 2

Identificador	Descripción	Prioridad
RF2ID1	El usuario puede obtener información sobre las peticiones sobre su modelo mediante el API	Media
RF2ID2	El sistema debe incluir la capacidad de creación de Colas de Mensajería para poder entrenar modelos máquinas predefinidas	Media
RF2ID3	El sistema debe implementar modelos V3 pre-determinados para entrenamiento	Media
RF2ID4	El usuario puede crear modelos Inception V3 a partir de imágenes predefinidas en Amazon S3	Media
RF2ID5	El usuario puede crear modelos Inception V3 a partir de imágenes predefinidas en cualquier medio	Media
RF2ID6	El usuario puede implementar diferentes modelos pre-determinados	Media
RF2ID7	El usuario puede implementar modelos personalizados	Media

Tabla 2: Requisitos Funcionales de la Fase 2

3.2. Requisitos no funcionales

Se presentan los requisitos no funcionales de la aplicación, según prioridad.

Identificador	Descripción	Prioridad
RNFD1	La aplicación WEB debe funcionar en todos los navegadores	Alta
RNFD2	Los usuarios pueden contactar con el soporte del Framework para proponer modelos	Media
RNFD3	El sistema debe incorporar un API CLI para testeo desde linea de comandos	Media
RNFD4	El framework debe ser tolerante a fallos	Alta

Tabla 3: Requisitos No Funcionales

4. Arquitectura

En este capítulo, se desarrollará la arquitectura del sistema. En primer lugar, se presentará una arquitectura global del sistema, en la que se interconectarán todos los sistemas implicados en el desarrollo del proyecto.

Y a continuación, se detallarán los tres grandes módulos implicados en el desarrollo de este proyecto, y el porqué de su inclusión.

4.1. Arquitectura global del sistema

El proyecto se divide en cuatro bloques, que corresponden a cada uno de los sistemas implicados en el proyecto, que interconectados, permiten el entrenamiento y gestión del Framework. Además, se añade un quinto bloque, que trata sobre la gestión de usuarios, y que se detallará mas adelante, en la capítulo de Diseño

En primer lugar, para poder acceder a la aplicación web, que permite dar de alta los modelos y usuarios, tienes que pertenecer a cualquiera de los grupos, y para poder realizar una gestión de los usuarios, así como de diferentes métricas, como pueda ser número de modelos por grupo, o número de roles por grupo, has de pertenecer al grupo ADMIN (Ver Anexo 2)

Por otro lado, al tratarse de una aplicación web creada en Angular 4, la componente de securización se realiza mediante token JWT, así como el servidor que atiende las peticiones dispone de un Rest API, además, las comunicaciones entre cliente y servidor se realizarán mediante el protocolo HTTPS.

Adicionalmente, para poder interconectar los módulos, se hace uso de colas de mensajería y contenedores Docker, permitiendo la portabilidad y escalabilidad del sistema en función de las necesidades del mismo.

El diagrama de la Figura 22, muestra como están los sistemas interconectados, y como las interactúan entre ellos, en función de la necesidad o requerimiento del usuario.

Sin mas dilación, una breve descripción de cada uno de los módulos que integran este proyecto:

- **Módulo WEB**: Se trata de la parte visible y de gestión del Framework, en él se pueden dar de alta usuarios, crear permisos y crear grupos, en el caso de tratarse de un usuario perteneciente al grupo de ADMIN, y en cualquiera de los casos, se podrán crear modelos, arrancarlos, pararlos y probarlos, en función de la necesidad y el grupo del usuario.

- **Servidor WEB**: Se encarga de atender todas las peticiones realizadas hacia él, tanto del módulo WEB, como del módulo API desde cualquier tipo de cliente, tanto JAVA como Swift. Todos los servicios que ofrece están securizados a alto y bajo nivel, mediante Spring Security y Token Based Authentication. Además implementa WebSockets para notificaciones dinámicas y se conecta con Amazon AWS para realizar la gestión de usuarios, así como la integración con la Base de Datos y la gestión de los contenedores Docker.

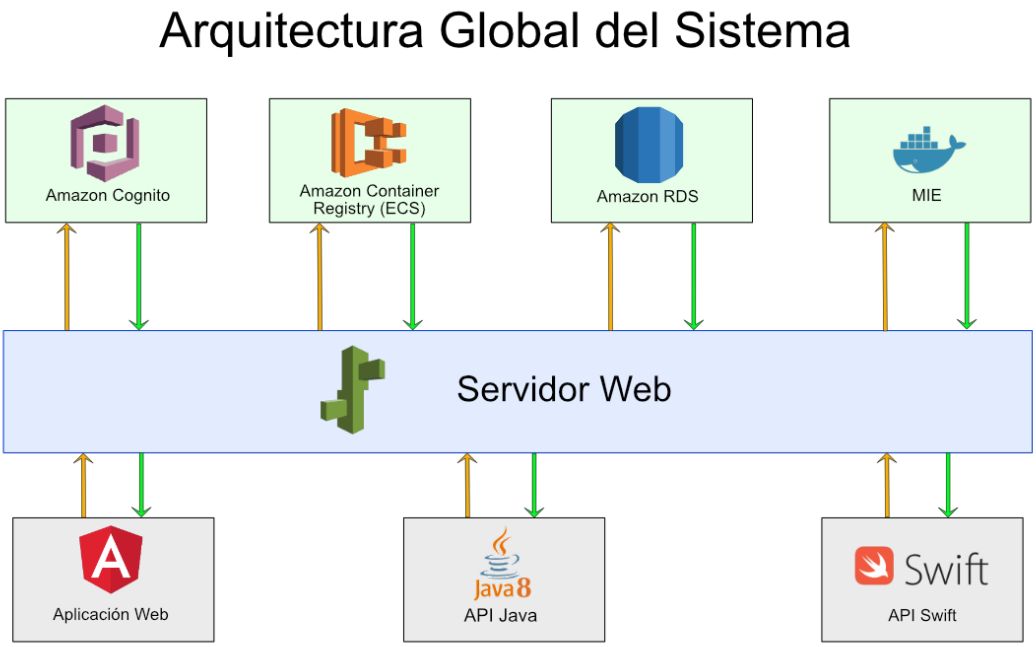

Figura 22: Arquitectura Global del Sistema

- **Módulo API**: Este módulo implementa un API para poder conectarse con cada uno de los modelos que cada usuario ha entrenado, y que tiene como implicación, el poder realizar predicciones en función del tipo de entrada que se introduzca, contra el modelo del usuario.
- **Módulo Integrado de Entrenamiento (MIE)**: Este módulo, que se ha denominado MIE, es un módulo que se instala, a través de Docker, en cada uno de los equipos con GPU capaces de entrenar modelos, y que está escuchando en una cola, esperando que alguien requiera que entrene un modelo, y ofrece feedback al servidor WEB con cada una de las actualizaciones del modelo, además, de contabilizar el tiempo de entrenamiento para poder computar esas horas en la monetización de la aplicación.

4.2. Arquitectura de la Aplicación Web

En este primer punto, se hablará de la arquitectura a nivel de Aplicación Web, que en el caso de este proyecto, se trata de una implementación en Angular 4, es decir, solamente la parte de vista y de lógica de negocio relacionada con la vista.

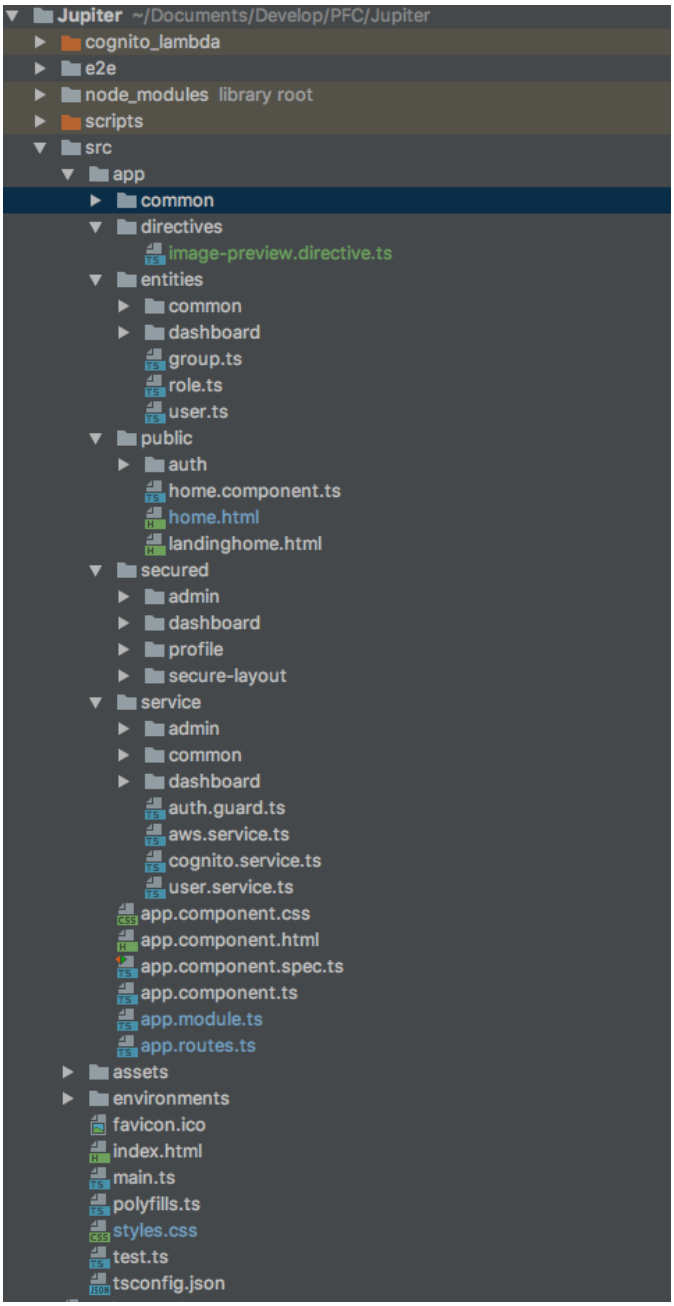

Figura 23: Estructura Proyecto Angular 4

Como se puede observar en la figura 23, la estructura del proyecto se organiza en varios módulos, relacionados por tipo de módulo y no por funcionalidad, o lo que es lo mismo, se agrupa mediante servicios, entidades, vistas securizadas, vistas publicas, etc...

La aplicación utiliza Angular Router para la implementación de las rutas de la

aplicación, así como para la securización de las vistas.

Implementa interceptores de peticiones HTTP, para introducir el token JWT, que permite realizar peticiones Rest Autenticadas, y de esta manera, se securiza el sistema de manera óptima.

Angular 4[15] , se trata de un framework completo, por lo que no se necesitan módulos adicionales a la hora de realizar peticiones, ni de implementar las vistas, ya que tiene su propio motor de plantillas, y no ha sido necesaria la implementación de un motor externo.

Cabe destacar, que se trata de un framework basado en componentes, lo que permite una modularización y reutilización de los mismos, lo que permite reducir los tiempos de desarrollo de cada una de las vistas y servicios.

Por último, es necesario indicar, que para facilitar la maquetación e implementación de las vistas de la aplicación, se ha utilizado Bulma.io[22] como Framework CSS, además de diversas utilidades para la subida de archivos, y alertas de aplicación.

4.3. Arquitectura del Servidor Web

Respecto a la arquitectura del servidor Web, se ha optado por una implementación basada en Spring 4[19], muy dinámica gracias a su módulo Spring Boot.

Como se puede observar en la figura 24, y al igual que en la arquitectura de la aplicación WEB, este módulo está estructurado por tipo de módulo y, no por funcionalidad.

Cabe destacar, que este módulo WEB es el puente entre todos los módulos e integradores que utiliza este proyecto, y por tanto, es necesario que esté operativo al 100 %.

Entre otras cosas, se encarga de conectar la aplicación WEB, con los servicios de autenticación y proporciona la autorización de los usuarios, además, se conecta con la base de datos, donde se encuentra toda la persistencia de datos del framework.

También es el encargado de la generación de modelos para cada usuario, de arrancarlos, pararlos, testearlos y poder eliminarlos. Además, es el encargado de escuchar peticiones del API Java y Swift, que permite realizar las predicciones desde el móvil o donde se desee, ya que conecta el API con la imagen Docker y el servidor GRPC.io[20] que se encarga de realizar la predicción según el modelo deseado.

Por tanto, es clara la importancia de este módulo respecto a la arquitectura global del sistema, ya que es el integrador de todos los demás, y por tanto, gracias a Spring Boot y Amazon, permite que sea escalable según la necesidad. Es más, este servidor WEB se puede llegar a comprender como si se tratase de un Bus de Servicios.

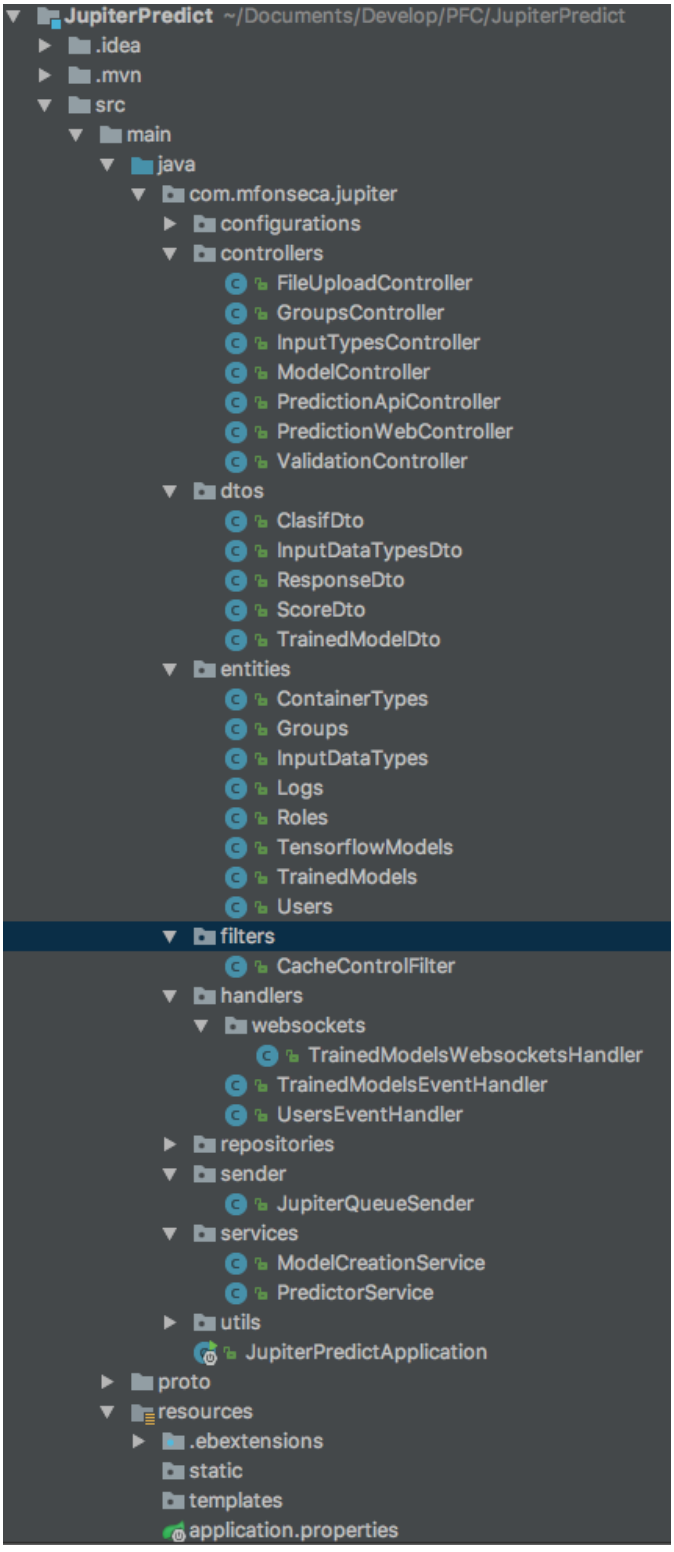

Figura 24: Estructura Proyecto Servidor WEB

4.4. Arquitectura del Flujo de Entrenamiento

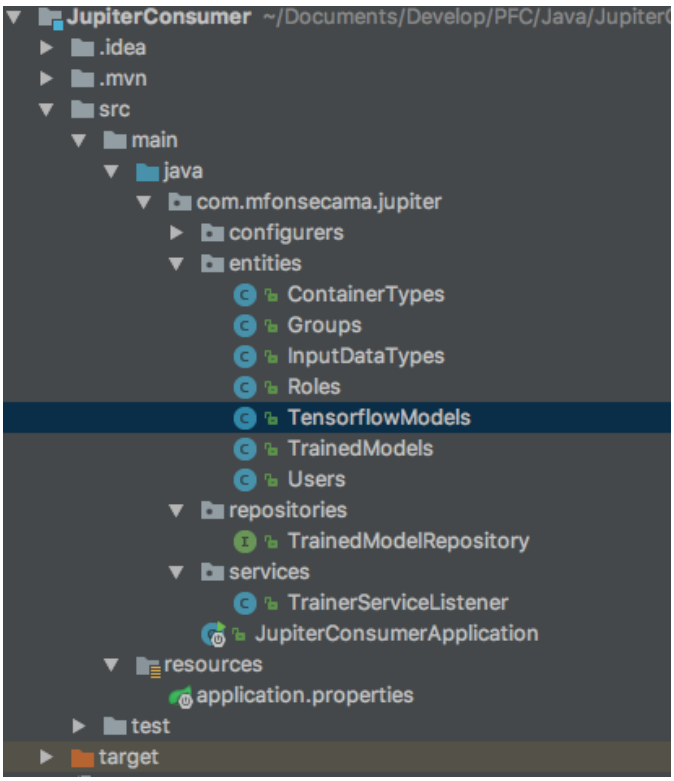

Figura 25: Estructura Proyecto Servidor WEB

Esta parte, se puede decir, que es la encargada del entrenamiento de modelos propios, según necesidades del usuario y atendiendo a unos requisitos pre-definidos, es decir, es donde entra en funcionamiento el módulo MIE, descrito en la arquitectura global del sistema.

A efectos prácticos, el servidor encola, en Amazon SQS[24], una petición de entrenamiento, previamente insertada en Base de Datos, y un cliente, que se ejecuta en Docker, atiende dicha petición en la máquina encargada de realizar el entrenamiento y la bloquea hasta que se complete, y en dicho proceso, va notificando al servidor, las actualizaciones en el modelo y contabilizando el tiempo de procesamiento necesario para el entrenamiento, permitiendo, más adelante, monetizar dicho tiempo.

Este cliente que se instala en Docker, se implementa con Spring 4 y Spring Boot, y se conecta a una cola, además, también tiene acceso la base de datos, ya que es necesario para las actualizaciones del modelo.

Una vez recibida la petición, con el API Java de Docker, se ejecuta en entrenador correspondiente al modelo indicado, y es éste entrenador modificado el que notifica al servidor WEB de las actualizaciones y tiempos de entrenamiento, que se almacenan en base de datos.

Para concluir es importante recalcar que no se trata de una arquitectura como tal, pero sí de una visión global de cómo está previsto que funcione módulo, cuya implementación está incluida en la Fase 2 de este proyecto.

4.5. Arquitectura de Autenticación y Autorización de Usuarios

En este último punto de la arquitectura, se explica cómo se ha de realizar la autenticación y autorización de usuarios, que se separa en dos módulos.

Para la autenticación se utiliza Amazon Cognito[27].

Según la definición de Amazon, "*Amazon Cognito le permite agregar el registro y el inicio de sesión de forma sencilla a sus aplicaciones web y móviles. Con Amazon Cognito, también tiene las opciones de autenticar a los usuarios a través de proveedores de identidad social, como Facebook, Twitter o Amazon, con soluciones de identidad SAML o a través de su propio sistema de identidad. Además, Amazon Cognito le permite guardar datos de forma local en los dispositivos de los usuarios, lo que permite que sus aplicaciones funcionen incluso cuando los dispositivos están fuera de línea*"

Es decir, un sistema automático de autorizaciones, lo que permite abstraer gran parte de la lógica del usuario a un proveedor externo.

Sin embargo, la parte de autorización sí la realiza el servidor WEB, es decir, qué puede ver y no puede ver el usuario, se almacena en la base de datos de la aplicación, en función de grupos y permisos, que se detallará mas adelante, en la parte de diseño.

5. Diseño

Este capítulo presentará el diseño completo de la primera fase del Proyecto Jupiter. Se propondrá un diseño del modelado de base de datos, así como de los diferentes elementos que conforman el Framework.

También se mostrará el diseño de las vistas de la Aplicación Web

5.1. Modelado de la Base de Datos

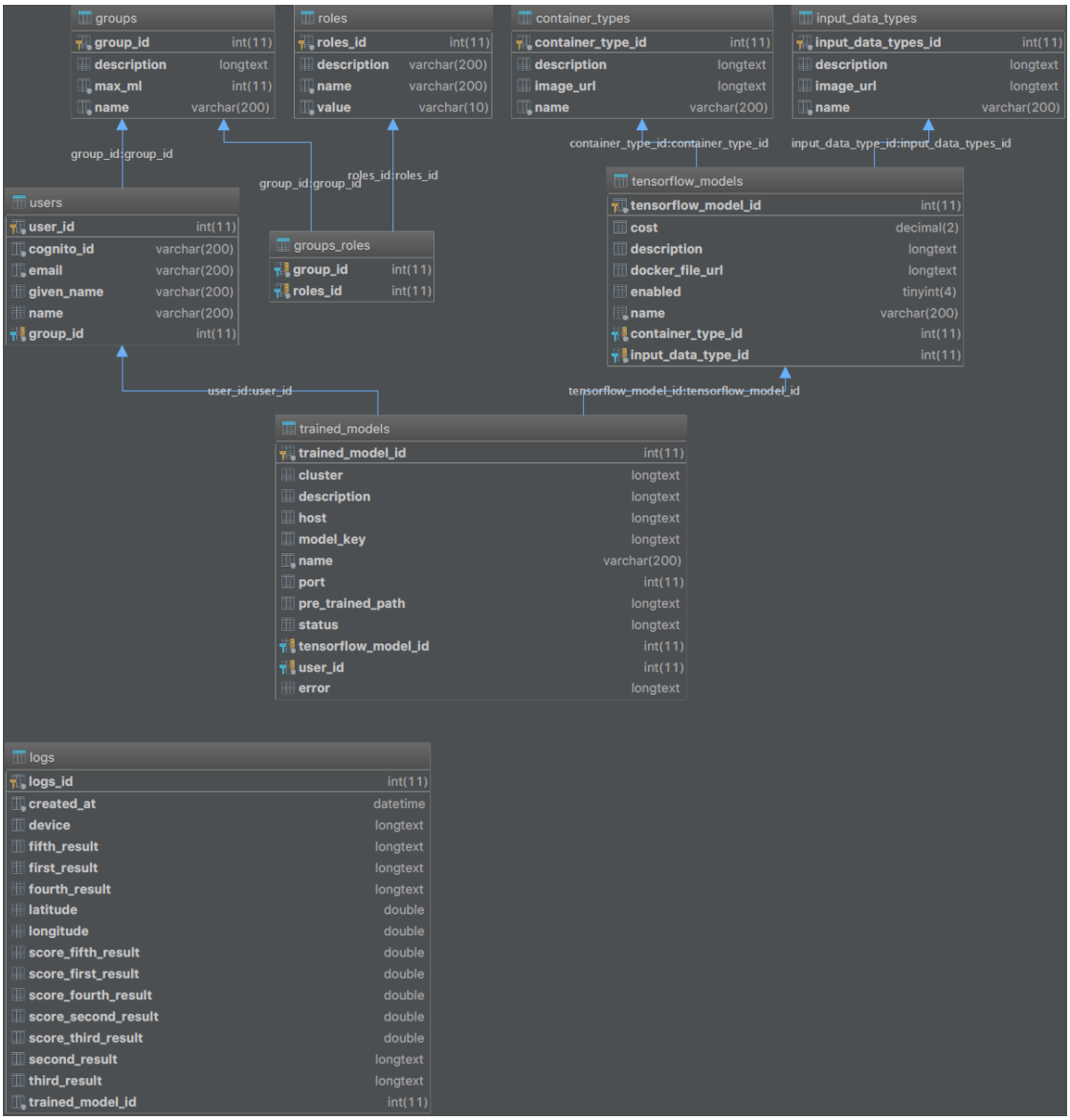

Figura 26: Modelado de Base de Datos

La creación de la Base de Datos se crea mediante la ejecución de un *Script* en Base de Datos, además, de venir acompañado por unos datos iniciales o *seeds* para el correcto funcionamiento de la aplicación (Ver Anexo 2)

A continuación la descripción de las entidades asociadas a la arquitectura del sistema (exceptuando el modelo de autenticación y autorización, que se describirá en el apartado a continuación):

- container_types: Contiene la información sobre los tipos de modelos admitidos, en la fase 1, el único modelo definido y permitido es Inception v3[28]
  - container_type_id: Identificador del contenedor
  - description: Descripción del contenedor
  - image_url: URL de imagen asociada a este tipo de contenedor
  - name: Nombre del tipo de contenedor
- input_data_types: Tipos de entradas admitidas para cada uno de los modelos. Por defecto se crean tres: Base de Datos, S3 y Pre-trained
  - input_data_types_id: Identificador del tipo de dato
  - description: Descripción del tipo de dato
  - image_url: URL de imagen asociada a este tipo de dato
  - name: Nombre del tipo de dato
- tensorflow_models: Modelos de tensorflow definidos, combinando la tabla de inputs y tipo de contenedor. Por defecto, solamente está habilitada Inception V3 con modelo pre-entrenado
  - tensorflow_model_id: Identificador del modelo tensorflow
  - cost: Coste de este modelo por hora de computación
  - image_url: URL de imagen asociada a este tipo de dato
  - name: Nombre del tipo de dato
  - docker_file_url: La URL donde se encuentra la imagen pre-definida para la realización de este entrenamiento
  - enabled: Si el modelo está habilitado o no
  - container_type_id: Asociación con la tabla container_types
  - input_data_type_id: Asociación co la tabla input_data_types
- trained_models: Modelos entrenados por un usuario en concreto, asociado a un tensorflow_model
  - trained_model_id: Identificador del modelo entrenado
  - cluster: Identificador del cluster en el que está el modelo preparado
  - description: Descripción del modelo dada por el usuario
  - host: Dirección de host en el cual está el modelo ejecutándose
  - model_key: Clave para el acceso al API del modelo
  - name: Nombre del modelo dado por el usuario

  - pre_trained_path: Si se trata de un modelo pre-entrenado, donde se encuentran esos checkpoints
  - status: Si está arrancándose, arrancado, parándose, parado, pendiente o con error.
  - tensorflow_model_id: Asociación con la tabla tensorflow_models
  - user_id: Asociación con la tabla user
  - error: Mensaje de error en el caso de error

- logs: Logs insertados para guardar los elementos que se han predicho en los diferentes modelos de los diferentes usuarios
  - logs_id: Identificador del log
  - created_at: Fecha de creación del log
  - device: Tipo de dispositivo desde el cual se hace la predicción
  - fifth_result: El quinto resultado predicho
  - first_result: El primer resultado predicho
  - fourth_result: El cuarto resultado predicho
  - latitude: La latitud GPS desde donde se ha realizado la predicción
  - longitude: La longitud GPS desde donde se ha realizado la predicción
  - score_fifth_result: Score del quinto resultado
  - score_first_result: Score del primer resultado
  - score_fourth_result: Score del cuarto resultado
  - score_second_result: Score del segundo resultado
  - score_third_result: Score del tercer resultado
  - second_result: El segundo resultado predicho
  - third_result: El tercer resultado predicho
  - trained_model_id: Asociación con la tabla trained_models

5.1.1. Diseño del modelo Autenticación - Autorización

Este subapartado, referente al diseño del modelo de Autenticación y Autorización, ha sido separado del apartado anterior, debido a que es un diseño implementable, de manera transparente, en cualquier aplicación que desee utilizar este modelo de Autenticación y Autorización con Amazon Cognito

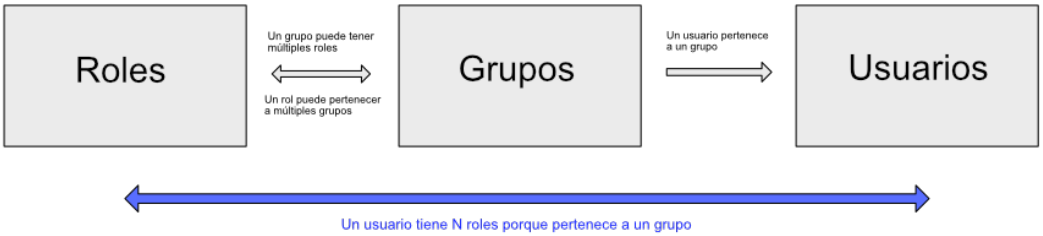

Figura 27: Estructura rol - grupo - usuario

Como se observa en la figura 27, se definen unos roles, que están asociados a los grupos que se deseen, y al usuario, lo que se le asigna es un grupo y no un rol directamente, lo que permite una gestión dinámica de roles y grupos, sin interferir en otros aspectos.

- roles: Contiene la información de todos los roles de la Aplicación
  - roles_id: Identificador del rol
  - description: Descripción del rol
  - name: Nombre del rol
  - value: Valores posibles que se quieran asignar al rol
- groups: Contiene la información de los grupos de la Aplicación
  - roles_id: Identificador del grupo
  - description: Descripción del grupo
  - name: Nombre del grupo
  - max_ml: Valores posibles que se quieran asignar al grupo
- groups_roles: Tabla de asociación entre grupos y roles
  - roles_id: Identificador del rol
  - groud_id: Identificador del grupo
- users: Tabla de usuarios
  - user_id: Identificador del usuario en base de datos
  - cognito_id: Identificador del usuario en Amazon Cognito
  - email: Email del usuario
  - given_name: Apellido del usuario
  - name: Nombre del usuario
  - group_id: Grupo al cual esta asociado el usuario

5.2. Diseño de los elementos que conforman el Framework

En este apartado, se explica el diseño a seguir de los contenedores, para agilizar el desarrollo de modelos según sean necesarios o requeridos.

5.2.1. Diseño de Contenedores Docker

Se diseña una estructura, que partiendo de un contenedor genérico de Ubuntu, se crea otro contenedor, denominado *generic_container*, cuyo objetivo es servir de base para el resto de contenedores con implementación de modelo específicos.

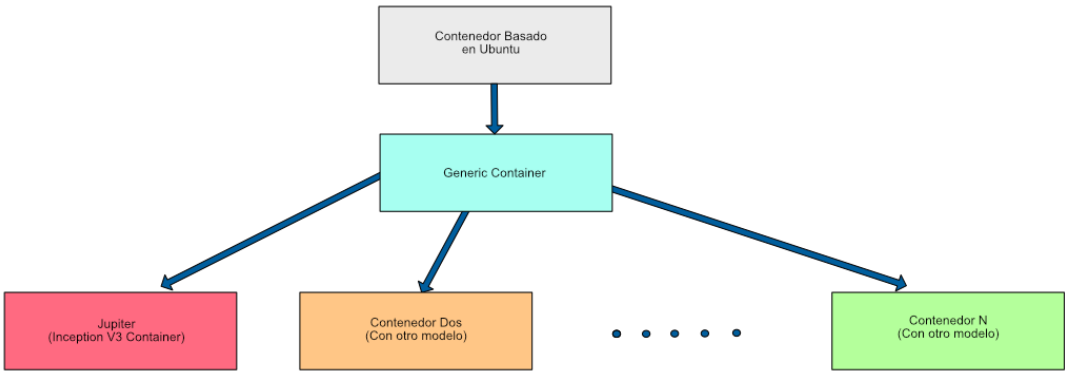

Figura 28: Estructura contenedores Docker

A partir de este modelo, se ha creado el siguiente contenedor:

- jupiter: Contiene el modelo Inception V3 pre-entrenado y está preparado, para que automaticamente, se ejecute con el modelo definido. Los siguientes parámetros han de ser pasados al contenedor para que se pueda ejecutar correctamente:
  - pre_trained_path: La url donde se encuentra el modelo pre-entrenado
  - port: El puerto expuesto para realizar las peticiones

5.3. Diseño de las vistas WEB

Página de bienvenida

Figura 29: Página de Bienvenida

En la Figura 29 se muestra la página de Bienvenida del Framework, es la página de inicio de la aplicación y donde se presentan las carácteristicas y posibilidades del Framework.

Login

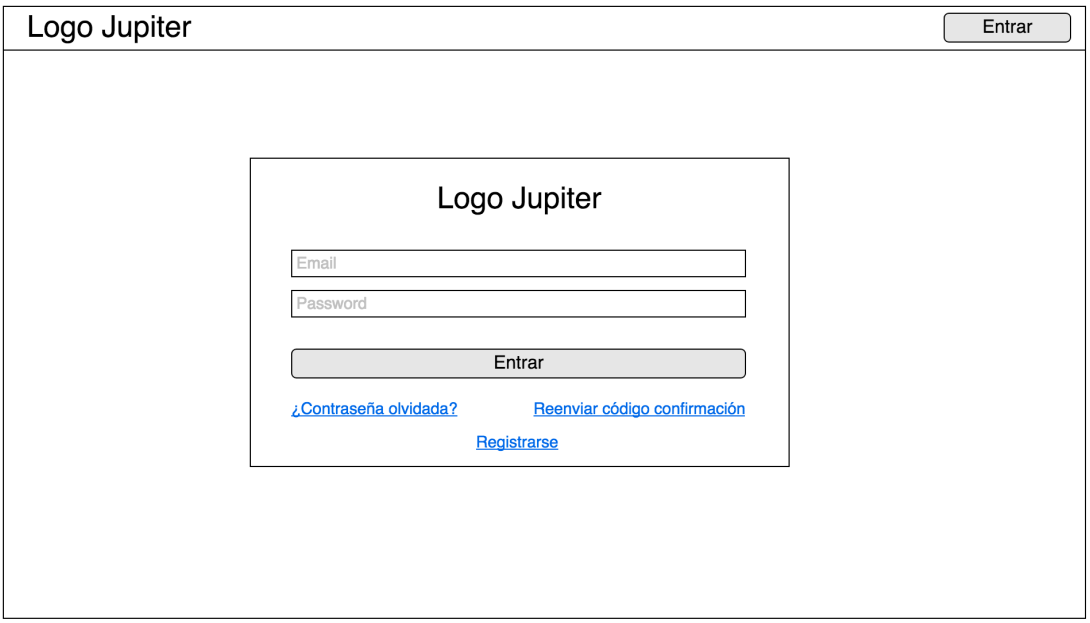

Figura 30: Página de Login

En la Figura 30 se muestra la página de Login del Framework, es decir, aquella en la que el usuario introducirá sus datos y le permitirá entrar en el Dashboard (31)

Dashboard

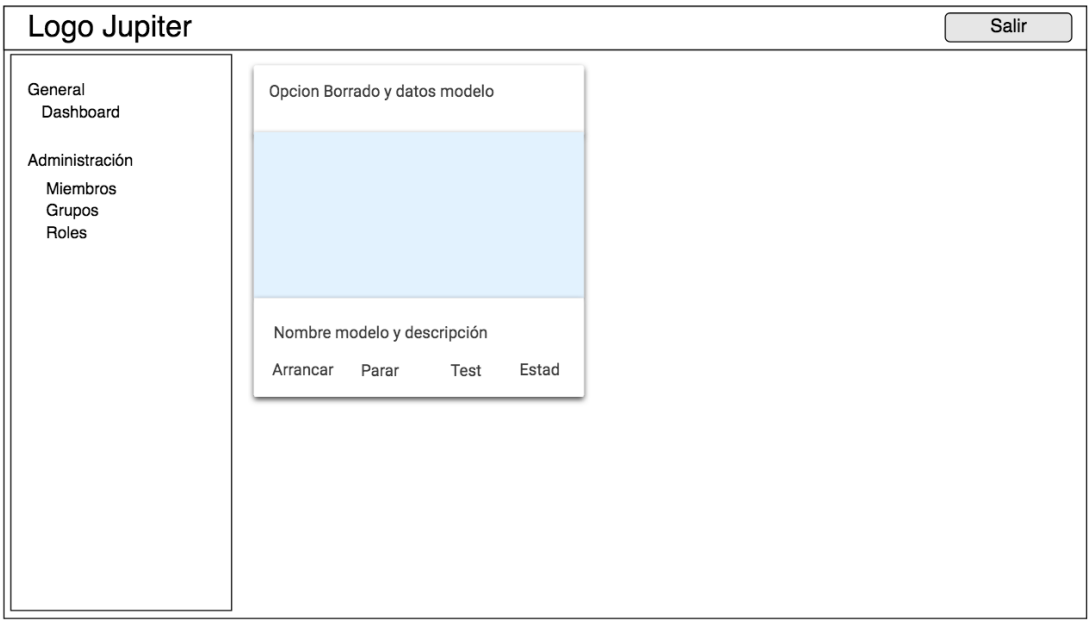

Figura 31: Pagina de Dashboard

En la Figura 31 se observa la página del Dashboard, que es aquella en la que aparecerán todos los modelos del usuario disponibles, y se podrán crear nuevos modelos si el grupo al que se pertenece lo permite.

Página de testeo

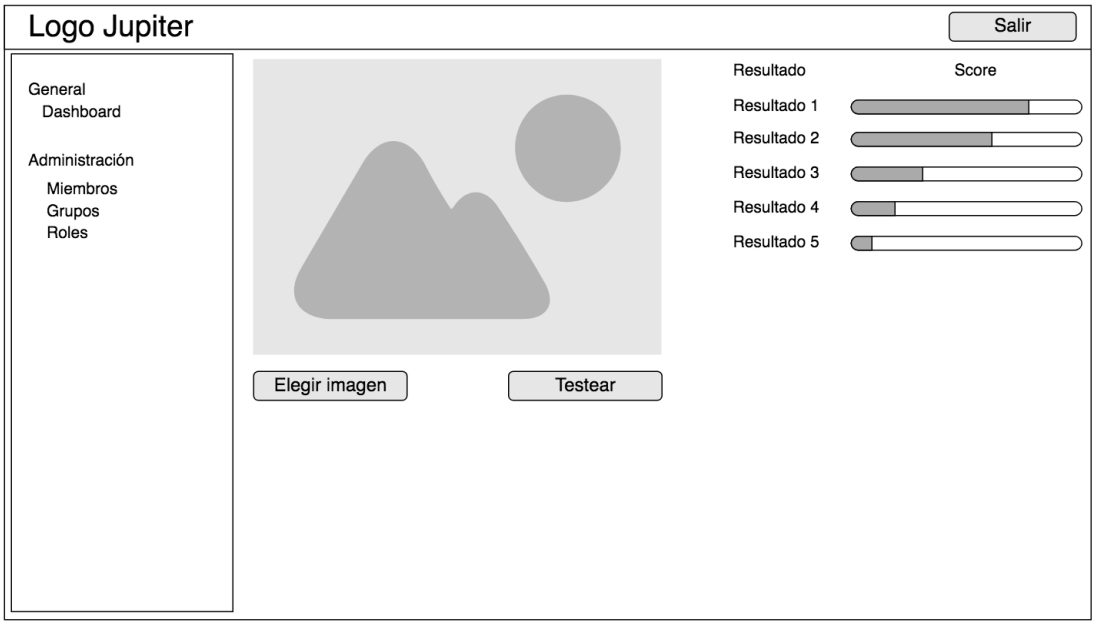

Figura 32: Página de Testeo

En la Figura 32 se observa la página de Testeo del modelo, se accede a través de cada uno de los modelos del Dashboard y permite realizar pruebas, subiendo imágenes y comprobando de que se tratan. En función del os modelos que se vayan implementando, aparte de imágenes, se podrán subir audios o textos.

Administración de miembros, grupos y roles

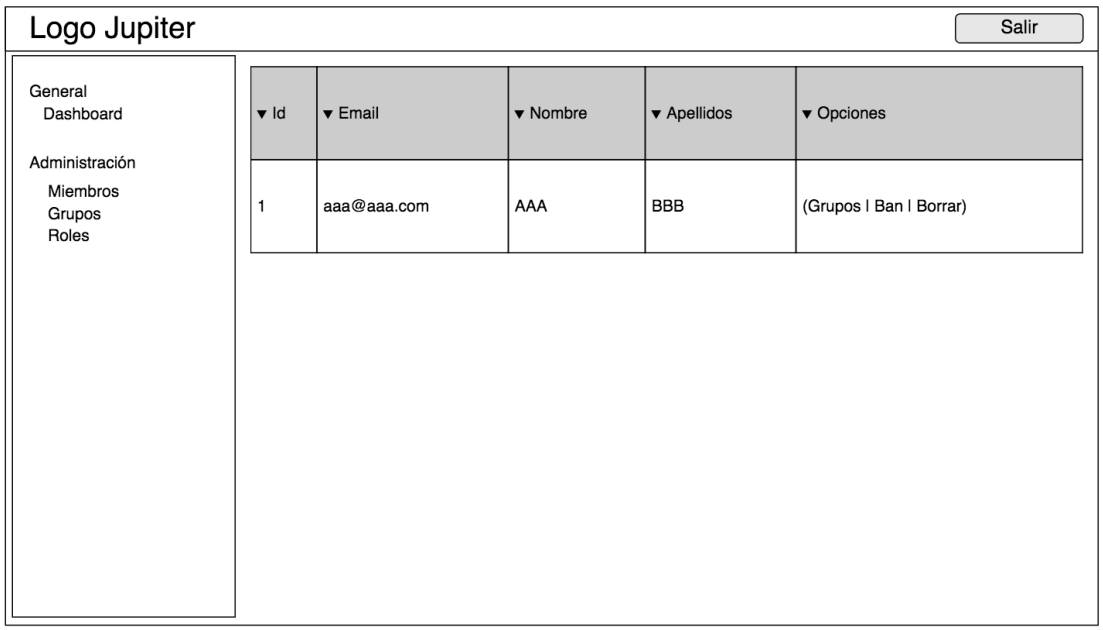

Figura 33: Página de Administración (1)

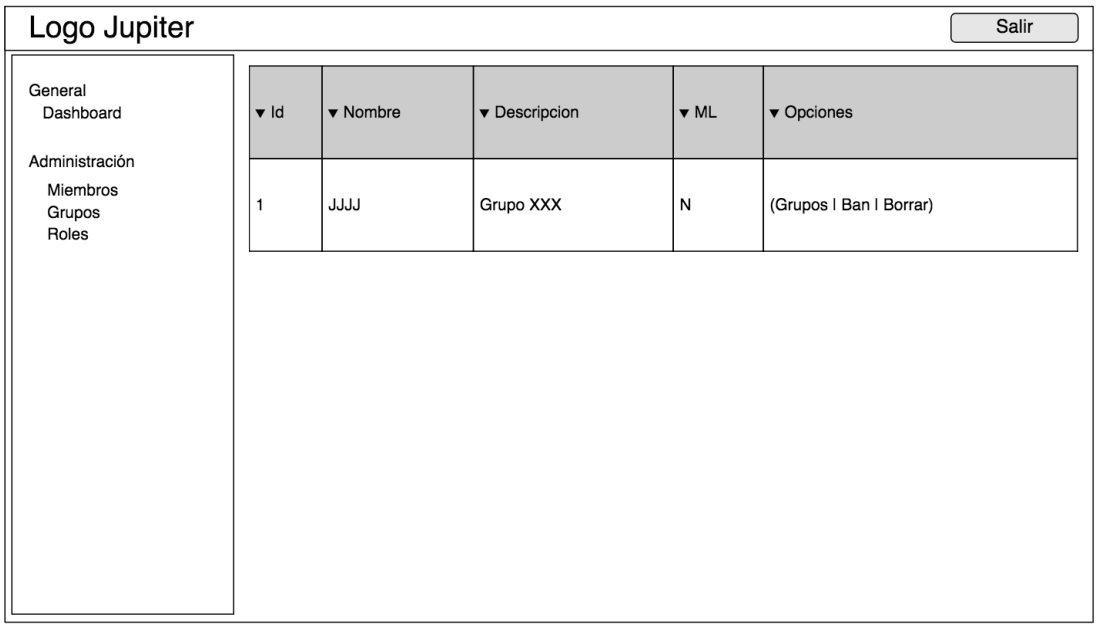

Figura 34: Página de Administración (2)

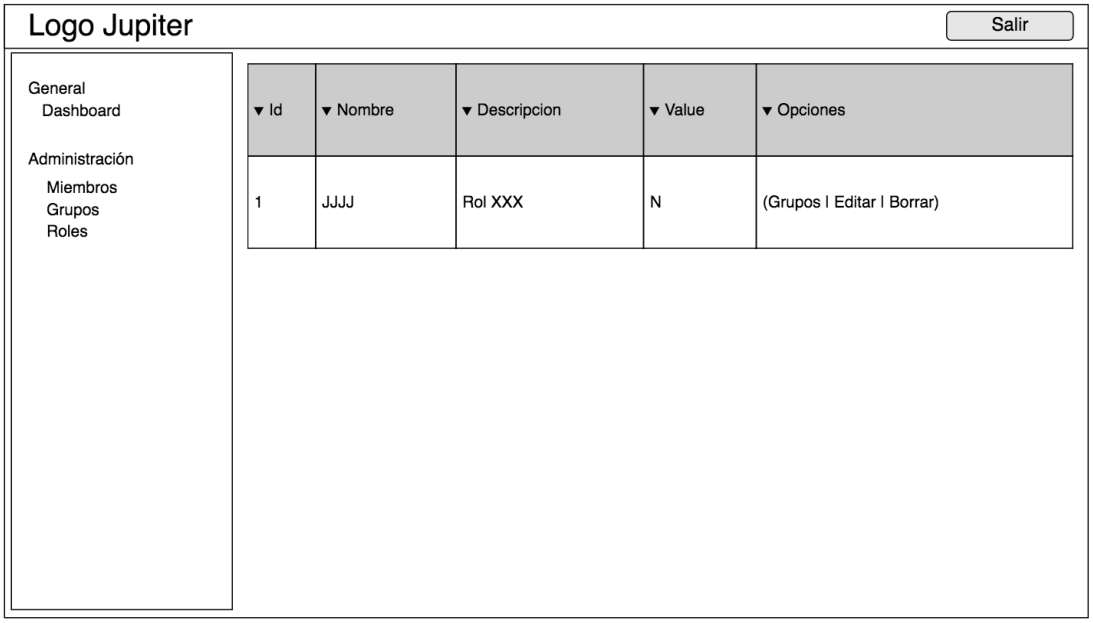

Figura 35: Página de Administración (3)

Las Figuras 33 34 y 35 muestran como serían las páginas del listado de usuarios, grupos y roles, respectivamente, de la parte de Administración de la aplicación. Se accede a ellas a través del menú lateral.

CRUD de roles (Extensible a miembros y grupos)

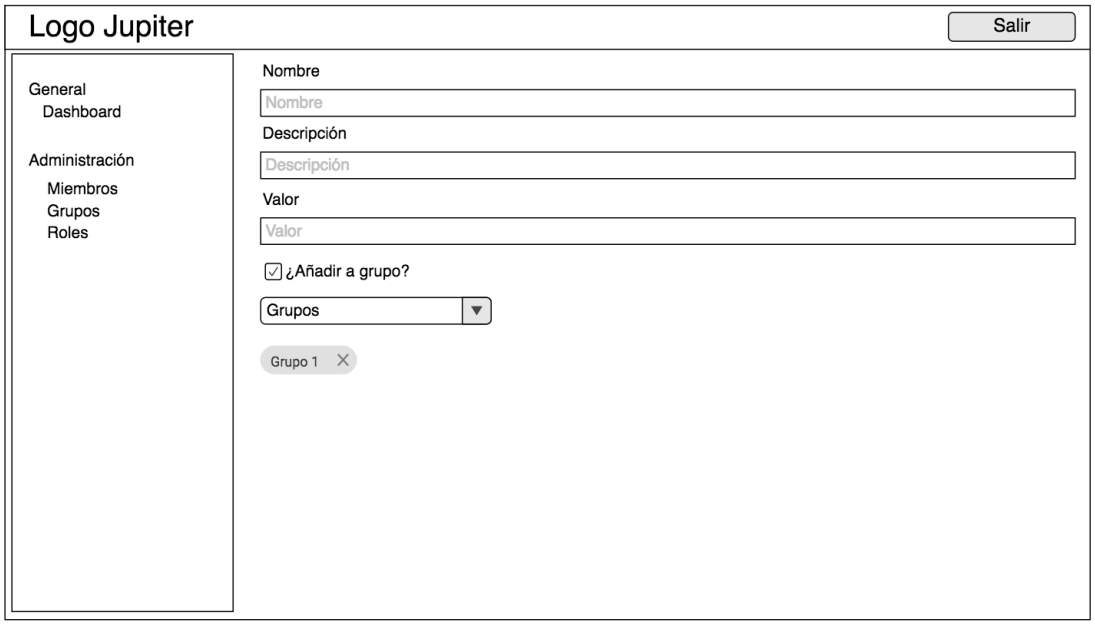

Figura 36: Página de CRUD

Por último, en la Figura 36 se muestra un ejemplo de cómo sería un CRUD de roles, en el que se editaría o crearía un rol, y se podría asignar a cualquier grupo.

6. Implementación

6.1. Introducción

Este capítulo presentará la implementación del Framework, así como de todos los componentes y otros Frameworks e servicios que interactúan con el mismo.

6.2. Integrándose con Amazon AWS

Como se explica en la arquitectura del sistema, así como en su diseño, para cumplir con los requisitos, se ha optado por la utilización de Amazon AWS como IaaS y PaaS, de esta manera, se permite una optimización de costes y tiempos, ya que la gestión del hardware y determinado software, está gestionado por ésta plataforma.

Esta una de las ventajas de usar proveedores externos, además, también te permite garantizar una disponibilidad muy alta de los servicios, y por tanto, del Framework a implementar.

Se ha dividido en secciones que explican como se ha implementado cada uno de los módulos que conforman el Framework.

6.2.1. Amazon Cognito

Se empezará hablando de como se ha implementado la Gestión de los Usuarios de la aplicación, ya que es Amazon Cognito es una de las claves que permiten realizar la autenticación de los mismos.

Figura 37: Vista Global Amazon Cognito

Como ya se comentó en apartados anteriores, la autenticación se realiza a partir de este proveedor externo, mientras que la autorización, si es la aplicación propia, mediante el módulo de Gestión de Usuarios.

En primer lugar, se crea un *pool de usuarios*, es decir, un grupo donde se inscribirán los usuarios, y gestionarán los mismos. Se pueden llegar a crear múltiples *pools* para cada funcionalidad de la aplicación, pero se opta por crear uno único, y gestionarlo por base de datos.

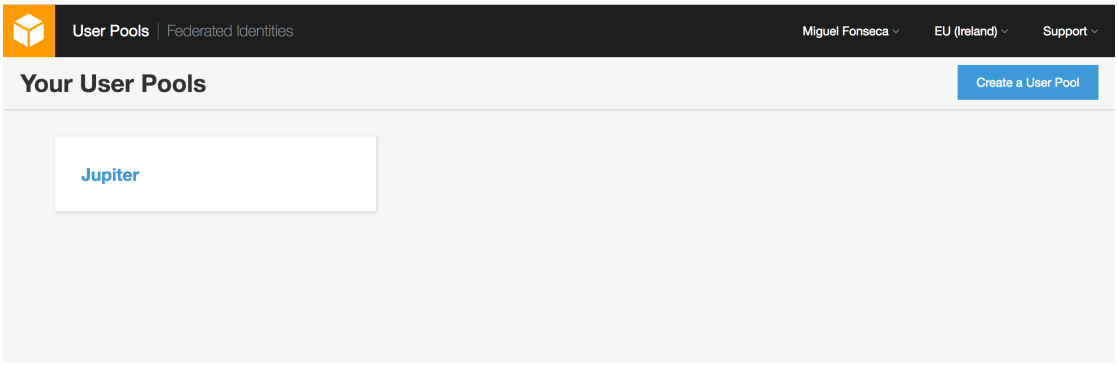

Figura 38: Dashboard de Jupiter Cognito

En la Figura 38 se observa como es el Dashboard en el cual se pueden crear los múltiples *pools* de usuarios, y cómo se ha indicado anteriormente, la creación de uno único para nuestra aplicación.

Entrando en detalle del *pool de usuarios* se ha de realizar una pequeña configuración, así como de una implementación, para poder usar la base de datos como gestor de autorizaciones de la aplicación.

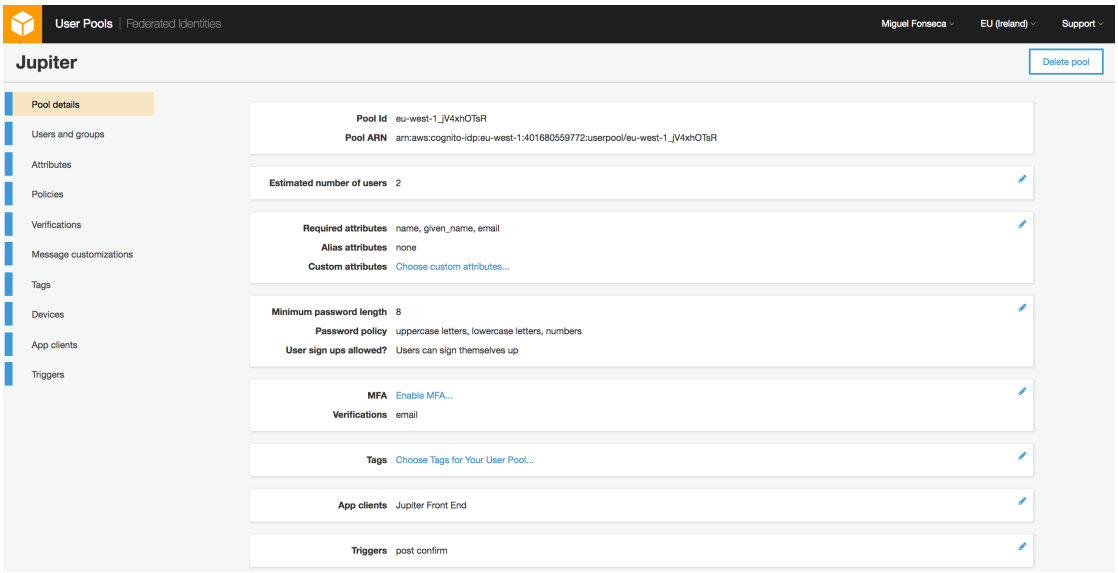

Figura 39: Resumen del *pool de usuarios* de Jupiter

En el resumen del *pool de usuarios Jupiter* (Figura 39), cabe señalar como importantes, los siguientes campos:

- Pool Id: Es el identificador único del pool de usuarios
- Pool ARN: Es el identificador interno para la interconexión de los múltiples servicios que provee Amazon AWS con este *pool*
- App clients : Las aplicaciones identificadas como aptas para usar este *pool de usuarios*, es decir, todas aquellas que no estén dadas de alta, no podrán usar

jamás este *pool*, y por tanto, securiza de qué manera los datos de nuestros usuarios.

- Triggers: Se detallará más adelante, pero es clave para poder enlazar la base de datos con Amazon Cognito

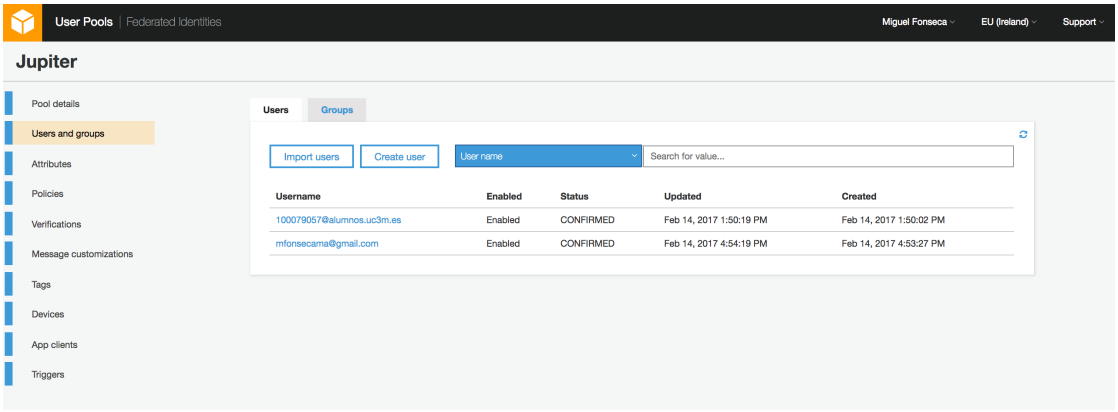

Figura 40: Usuarios inscritos en Jupiter

A continuación (Figura 40), se observa la característica de Amazon Cognito para la gestión de los usuarios, y se observan dos pestañas, Usuarios y Grupos. En un principio, se pensó la utilización de Grupos para gestionar las autorizaciones del Framework, pero finalmente, se optó por lo usarlo, por la inflexibilidad del mismo. Se observa como también existe la posibilidad de importar usuarios, de crearlos y de editarlos si es necesario, así como de borrarlos o desautorizarles el acceso.

Amazon Cognito incorpora un sistema de Autenticación completo (Figura 41), es decir, permite la verificación mediante emails, de mensajes de invitación e incluso una autenticación en dos pasos, con códigos de verificación, lo cual, es óptimo para la securización, y es una posible mejora de cara al futuro.

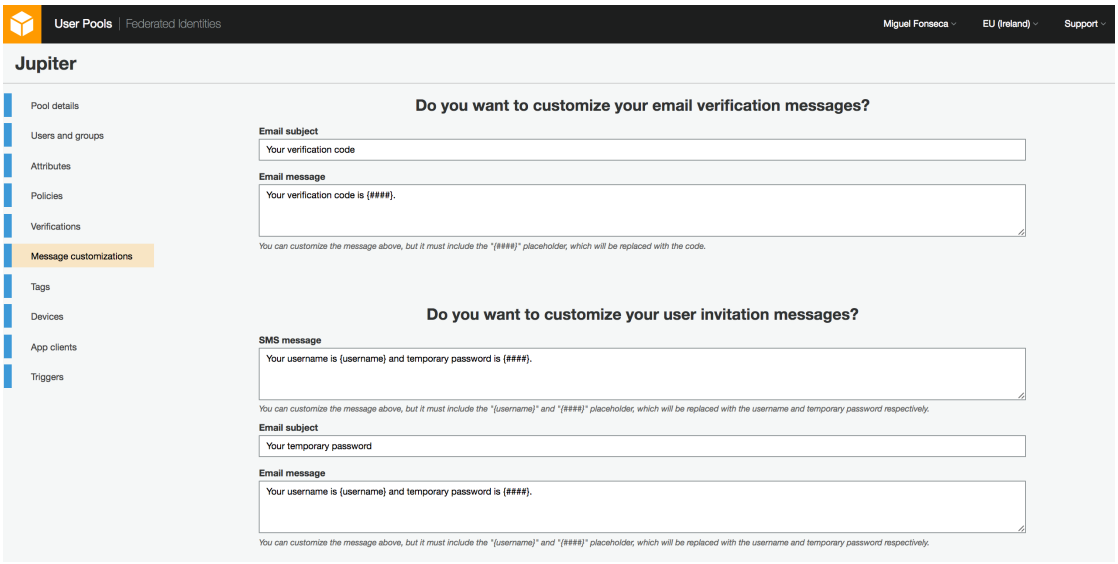

Figura 41: Verificación de Amazon Cognito

Como se comentaba anteriormente, permite restringir el acceso a las aplicaciones que el arquitecto o la empresa considere, de tal manera, que añade un grado más a la seguridad del sistema. (Figura 42)

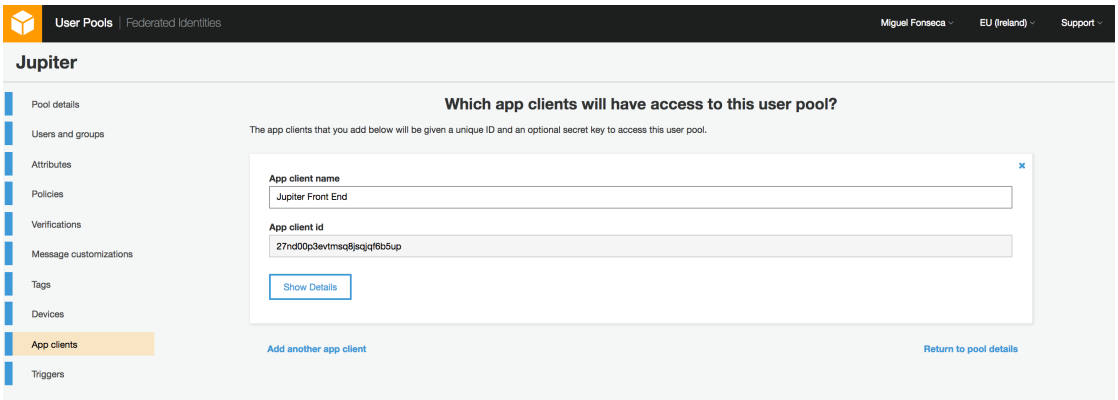

Figura 42: Aplicaciones autorizadas Jupiter

Por último destacar la opción de *Triggers*, donde se encuentra una de las claves de nuestra aplicación, ya que permite lanzar eventos en función de las operaciones que se hayan realizado sobre Amazon Cognito. (Figura 43)

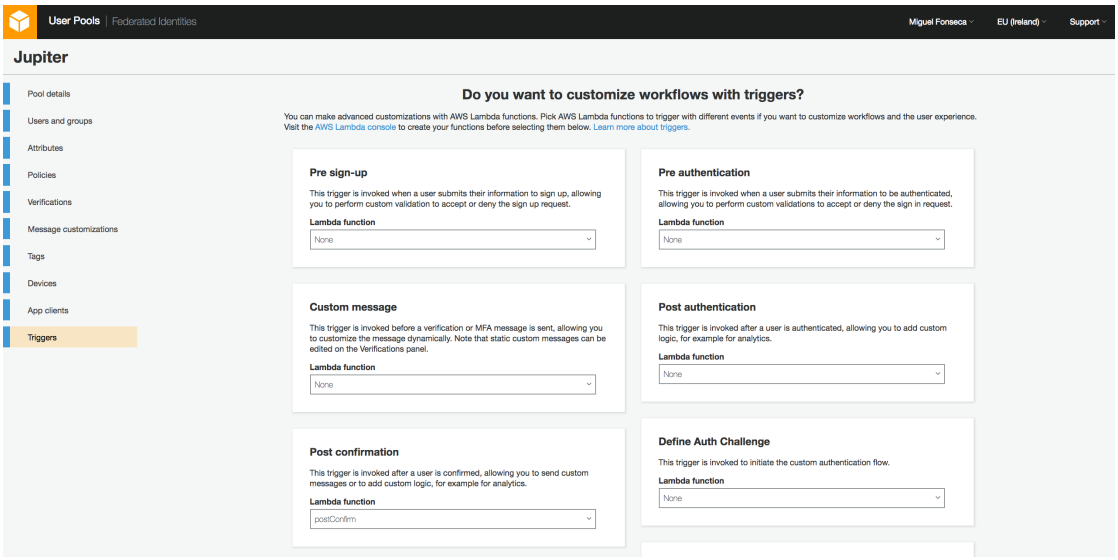

Figura 43: Triggers Jupiter

En el caso que nos ocupa, el único *trigger* que se implementa es el de post-confirmación, tras registrarse en la aplicación, y se encarga de insertar en base de datos, los datos del usuario registrado, así como, la referencia del mismo en el *pool de usuarios*.

Para ello, se utilizan AWS Lambda. Se puede describir AWS Lambda como funciones cortas, que reciben unas entradas y proporcionan unas salidas y que son ejecutadas a voluntad o por otras aplicaciones. La implementación es muy básica, ya que simplemente, realiza una inserción en BBDD, el siguiente fragmento de código muestra la implementación del mismo.

```
1
2  var mysql = require('mysql');
3
4  var connection = mysql.createConnection({
5  host     : 'xxxxxxx',
6  user     : 'user',
7  password : 'pass',
8  port     : '3306',
9  database : 'jupiter_web'
10 });
11
12 exports.handler = (event, context, callback) => {
13
14 console.log(event);
15
16 var username = event.userName;
17 var email = event.request.userAttributes.email;
18 var sub = event.request.userAttributes.sub;
19 var given_name = event.request.userAttributes.given_name;
20 var name = event.request.userAttributes.name;
```

```
21
22  var toInsert = {
23  'cognito_id' : sub,
24  'email' : email,
25  'given_name' : given_name,
26  'name' : name,
27  'group_id' : 2 //JUNO
28  };
29
30  console.log(toInsert);
31
32  connection.query("INSERT INTO users SET ?", toInsert,
       function (error, results, fields) {
33
34  console.log(error);
35  console.log(results);
36  console.log(fields);
37
38  callback(null, event);
39  });
40  };
```

6.2.2. Amazon RDS

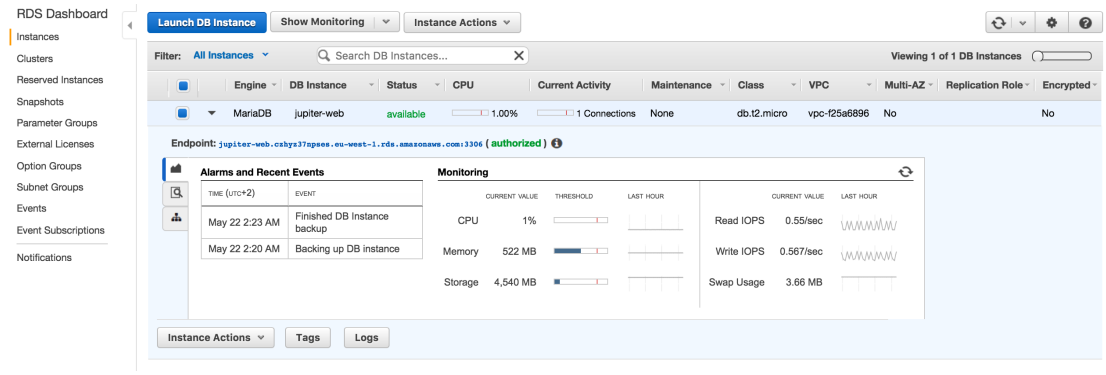

Figura 44: Instancias de Base de Datos

En esta sección, se explican como se ha implementado la Base de Datos sobre Amazon RDS. Para ello, se ha creado una instancia de la misma, utilizando como base de datos relacional, MariaDB[29], en la cual se han cargado los scripts descritos. (Ver Anexo 1)

Para ejecutar dichos scripts, el desarrollador se puede conectar a la base de datos mediante su cadena de conexión a través del terminal o de una aplicación, como pueda ser *MySQL Workbench*, o *Datagrip*. Una vez conectado, se ejecuta el script mediante el comando correspondiente o copiando dicho código.

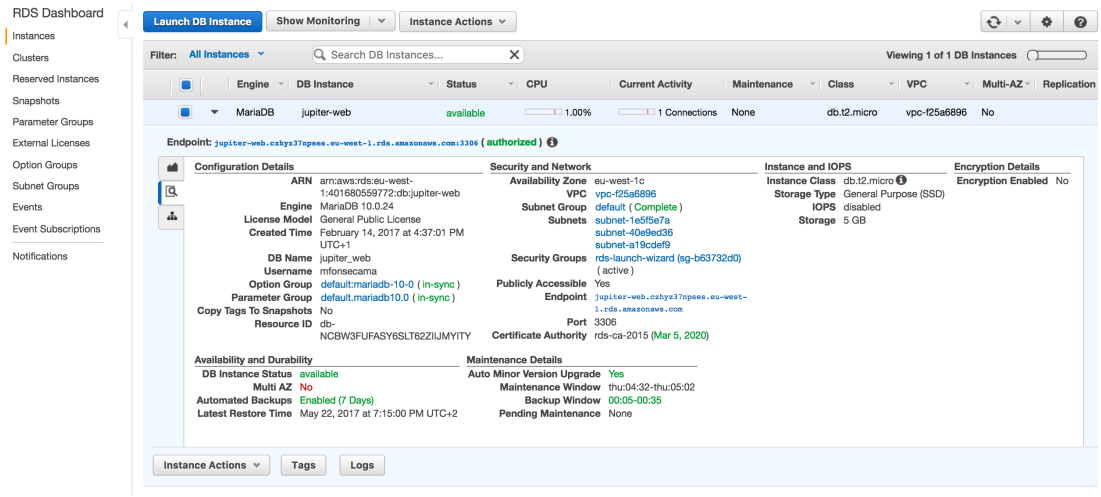

Figura 45: Datos de nuestra Base de Datos

Una de las características mas importantes que ofrece Amazon RDS respecto a cualquier otro proveedor, es la posibilidad de clusterización de la base de datos, en caso que se necesite, así como de una monitorización importante, permitiendo la detección de fallos y accesos en el caso que sea necesario.

6.2.3. Amazon S3

Amazon S3 es la tecnología de almacenamiento físico de datos que utilizará el Framework para almacenar todos los modelos, todas las imágenes Docker y todo aquello que sea necesario para el correcto funcionamiento del mismo.

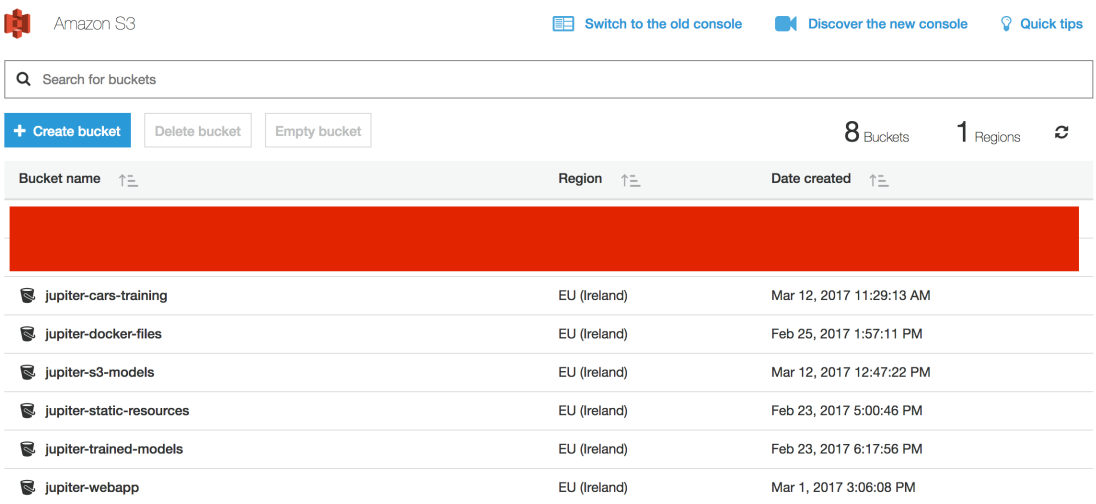

Figura 46: Buckets utilizados en Jupiter

Se divide en Buckets (Figura 46), es decir, contenedores de datos, y cada uno de ellos, tendrá una función específica:

- jupiter-cars-training: Son datos de entrenamiento para modelos de prueba
- jupiter-docker-files: Este bucket se encarga de almacenar un backup de todas las imágenes Docker que se usan o se usarán en la aplicación.
- jupiter-s3-models : Este bucket es el encargado de almacenar los datos de entrenamiento que ya están preparados, organizados y modificados para poder iniciar el entrenamiento en cuanto un MIE quede libre.
- jupiter-static-resources: Recursos estáticos para alimentar tanto las aplicaciones web como las aplicaciones móviles (imágenes, fuentes, etc...)
- jupiter-trained-models: Este bucket almacena los modelos ya entrenados, y que están disponibles para usar en las predicciones.
- jupiter-webapp: La aplicación web del Framework

Cabe señalar de la lista anterior, el último punto, en que se observa que la aplicación web se encuentra en un Bucket. Ésto es debido a que S3 también permite servir el front-end de cualquier aplicación web. En nuestro caso, al tratarse de una aplicación escrita en Angular 4, se puede servir desde éste contenedor.

6.2.4. Amazon ElasticBeanStalk

Otra de las piezas claves para la implementación del Framework. En EBS (ElasticBean Stalk) se ejecuta nuestro servidor WEB.

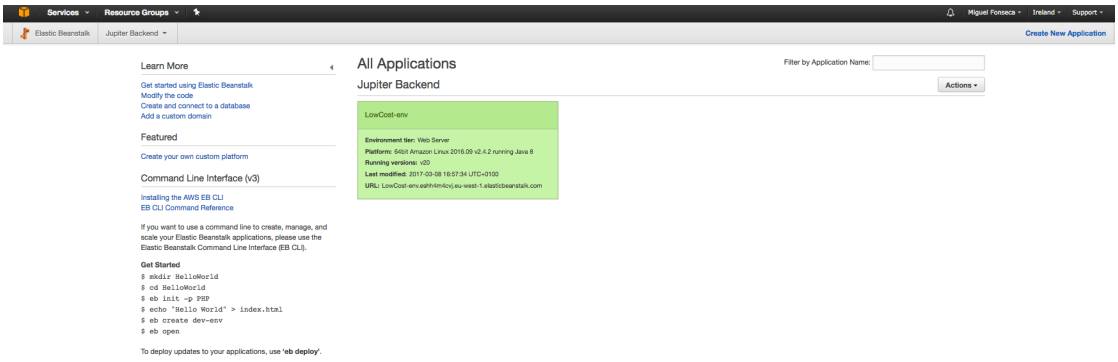

Figura 47: Dashboard EBS

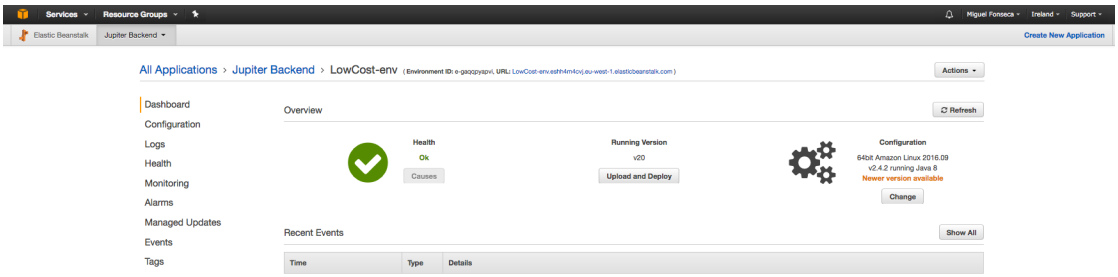

Figura 48: Dashboard Jupiter Backend

EBS es un contenedor de aplicaciones, como pueda ser un Apache Tomcat, completamente configurable, escalable y monitorizable, lo cual, es indispensable para cualquier backend que se precie.

Como se observa en la Figura 48, se puede comprobar la salud de la aplicación (Si tiene red, si hay errores de memoria, etc), así como ver la versión que está desplegada y la configuración de la máquina sobre la cual se está ejecutando.

En la pestaña de Configuración (Figura 49), se puede indicar si se quiere escalar la aplicación, la configuración del a instancia o la máquina virtual, la salud y la configuración del software (variables de entorno, principalmente)

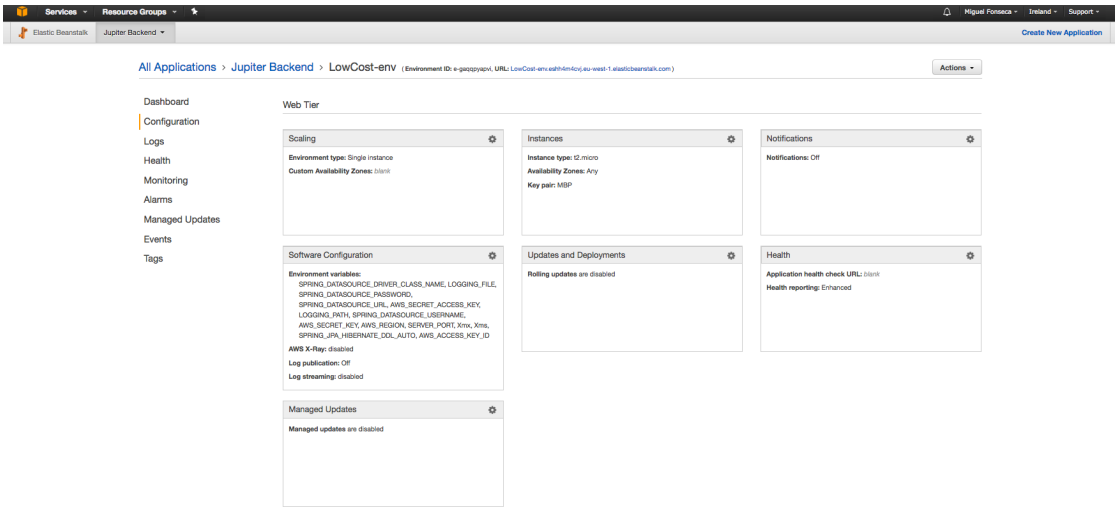

Figura 49: Configuración Jupiter Backend

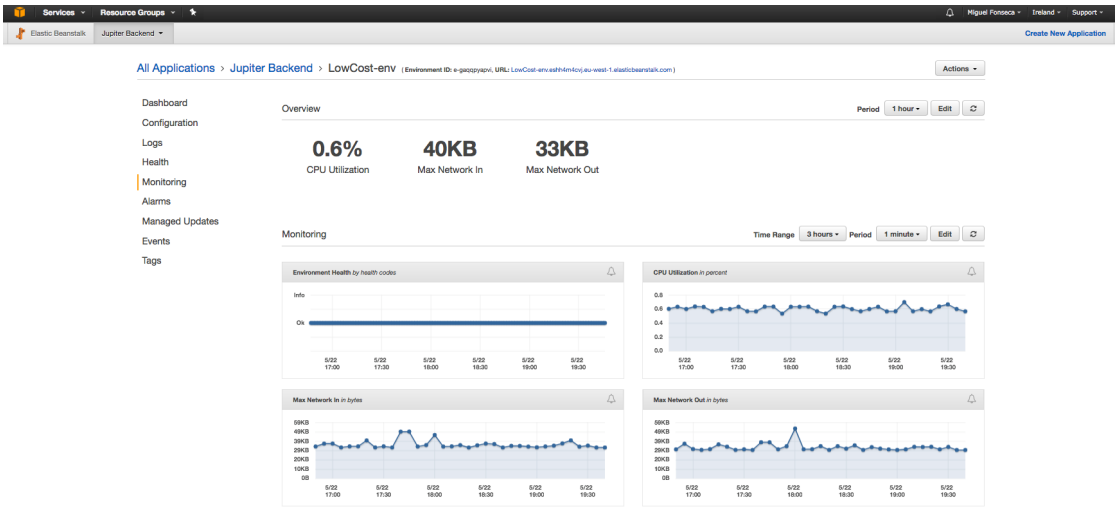

Figura 50: Monitorización Jupiter Backend

Es importante destacar, la pestaña de Monitorización (Figura 50), en la cual aparece la carga que tiene la aplicación, tanto de red como de CPU, mediante la cual, se podrán ver peticiones extrañas, sobrecargas, ataques DoS, y tomar decisiones de negocio, como escalar o mejorar la configuración del backend

6.2.5. Amazon Container Service

Este módulo es el encargado de almacenar los repositorios de nuestras imágenes Docker, es decir, guarda las imágenes Docker que se utilizarán para crear múltiples réplicas (Contenedores Docker) en función de la necesidad de los modelos que desee usar un consumidor del Framework.

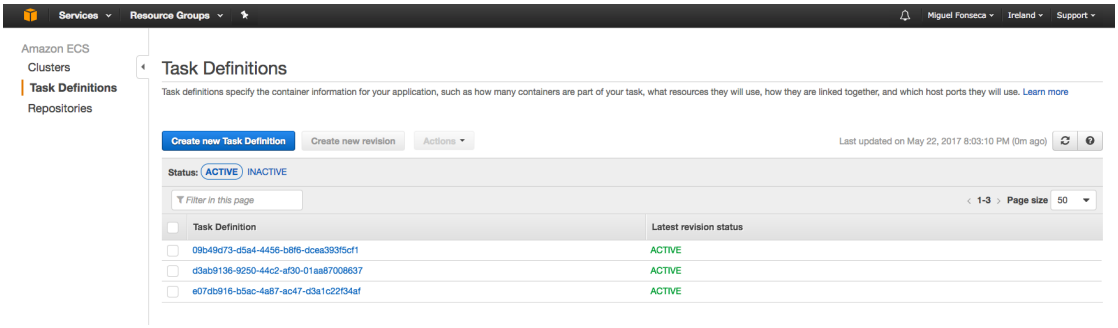

Figura 51: Task Definitions Jupiter

Figura 52: Repositorios de imágenes Docker de Jupiter

Estas dos funcionalidades se dividen en dos pestañas, las cuales son explicadas a continuación:

- Repositorios (Figura 52): Los repositorios es el lugar donde se almacenan las imágenes Docker, y que permiten ser replicadas con los parámetros que se desee en diferentes clusters o máquinas virtuales, mediante las definiciones de tareas.

- Definiciones de tareas o Task Definitions (Figura 51): Cada contenedor Docker que se ejecuta, tiene asociada una tarea. Se puede definir una tarea, como un servicio encargado de gestionar los estados del contenedor Docker (arranque y parada principalmente)

6.2.6. Amazon SQS

Se termina la sección de integración con Amazon SQS, o lo que es lo mismo, el gestor de colas de mensajería de Amazon, y se utiliza como medio de comunicación entre el Servidor Web y los MIE.

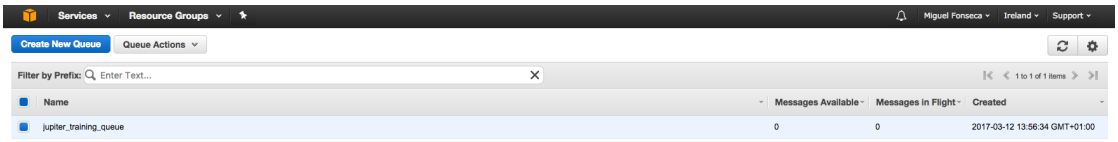

Figura 53: Cola de entrenamiento de Jupiter

Para ello, se ha definido una cola de entrenamiento, llamada *jupiter_training_queue* (Figura 53), en la cual se encolarán los modelos que están pendientes de entrenar, y será el MIE el que escuchando dicha cola, empiece a entrenarlos.

6.3. Aplicación WEB

En esta sección se desarrollará como se ha implementado la parte front-end de la aplicación web de gestión para los usuarios y administradores de Framework.

Se ha dividido en diferentes apartados, en los cuales se definirán las características mas importantes de cada uno de ellos.

6.3.1. Definición de rutas

Respecto a la definición de rutas, están separadas en dos grandes grupos: Rutas públicas y rutas seguras.

Las rutas públicas son aquellas en las que no es necesario estar autenticado en la aplicación web para poder acceder a ellas, y son las siguientes:

- **home**: La página de inicio de la aplicación web
- **about**: Ruta que tiene datos sobre la empresa y el Framework
- **login**: Ruta que dirige a los usuarios a la pantalla de Login
- **register**: Ruta que dirige a los usuarios a la pantalla de Registro
- **confirmRegistration/:username**: Ruta con un parámetro que permite dirigir a un usuario a la pantalla de confirmación del registro del mismo.
- **resendCode**: Ruta que dirige a un usuario a la pantalla de reenvio de código para confirmar el registro
- **forgotPassword**: Ruta que permite a un usuario introducir sus datos para poder recuperar la contraseña
- **forgotPassword/:email**: Segundo paso asociado a la ruta anterior.

Por otro lado, se encuentran las rutas seguras, es decir, aquellas en las cuales tienes que estar autenticado en la aplicación, y además, algunas de éstas requieren pertenecer a cierto grupo dentro de la aplicación, como puedan ser las rutas de administración. Son las indicadas a continuación:

- **logout**: Ruta que permite realizar el logout de la aplicación.
- **dashboard**: Ruta que dirige a la pantalla de inicio del usuario, donde se encuentran sus modelos.
- **model/wizard/create**: Ruta que dirige hacia la creación de un nuevo modelo de predicción
- **model/test/:id**: Ruta que permite testear un modelo con un id determinado.
- **members** (Solo **ADMIN**): Ruta que permite llegar a la página donde se listan todos los miembros. registrados

- **members/create** (Solo **ADMIN**): Ruta que dirige a la página de creación de un nuevo miembro de manera manual, sin que éste tenga que registrarse.
- **members/update/:id** (Solo **ADMIN**): Ruta que permite actualizar un miembro
- **groups** (Solo **ADMIN**): Ruta que permite llegar a la página donde se listan todos los grupos creados.
- **groups/create** (Solo **ADMIN**): Ruta que permite crear un grupo
- **groups/update/:id** (Solo **ADMIN**): Ruta que dirige hacia la actualización del grupo indicado
- **roles** (Solo **ADMIN**): Ruta que permite llegar a la página donde se listan todos roles creados.
- **roles/create** (Solo **ADMIN**): Permite la creación de nuevos roles
- **roles/update/:id** (Solo **ADMIN**): Permite la actualización de nuevos roles

6.3.2. Autenticación y autorización

En este apartado, se define y explica como se ha implementado, el flujo de Autenticación y Autorización de la aplicación, pieza clave para el correcto funcionamiento y la protección de los datos de los usuarios.

Figura 54: Pantalla de Login de Jupiter

En primer lugar, es importante entender el flujo de la Autenticación y Autorización, y todos los componentes que intervienen en ella.

En la Figura 54 se muestra la pantalla de Login de la aplicación, en la cual el usuario, introduce el email con el cual se ha registrado y la contraseña, y al pulsar en *Entrar*, se accede a la aplicación siempre y cuando se le autorice.

Para ello, lo primero que hace la aplicación web es comprobar en Amazon Cognito si el usuario está registrado, y si sus datos son correctos. En caso

afirmativo, con la respuesta, realiza una petición al servidor Web, mediante la cual, obtiene el grupo al que pertenece el usuario, con sus respectivos roles.

Una vez cargados dichos roles, continuará hacia la pantalla Dashboard (Figura 66)

Aparte del login de la aplicación, existe la posibilidad de registrarse en la misma (Figura 55). Una vez introducidos todos los datos obligatorios, se accede a la pantalla de confirmación, en la cual se ha de introducir un código que se envía al correo. (Figura 56). Cuando se ha introducido dicho código, se considera que se ha registrado en la aplicación, se guarda el usuario en la BBDD y se le asigna el grupo por defecto, JUNO.

Figura 55: Registro de Jupiter

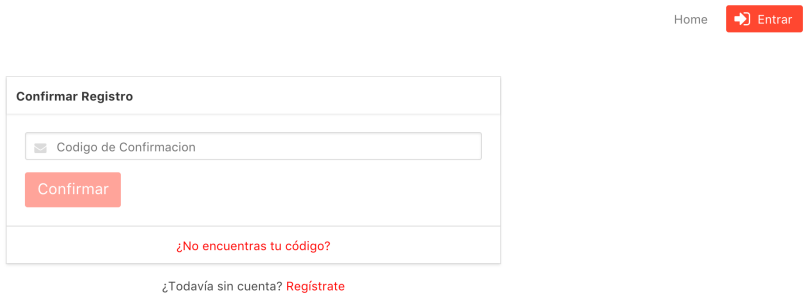

Figura 56: Confirmación del registro de Jupiter

Por último, existe la posibilidad de reenviar el código, en el caso de que el paso de confirmación se pueda realizar en ese momento, y la posibilidad de recuperar la contraseña en el caso de que se haya olvidado. (Figuras 57 y 58)

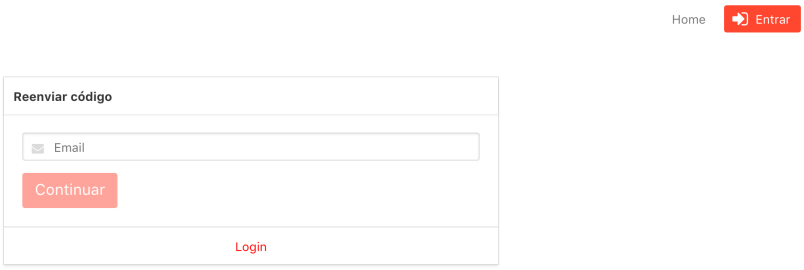

Figura 57: Reenvío de código de confirmación de Jupiter

Figura 58: Formulario de contraseña olvidada en Jupiter

6.3.3. Servicios

Bien es sabido, que para poder comunicarse con el servidor y poder realizar toda la lógica de negocio de una aplicación es necesario la implementación de servicios. (Figura 59)

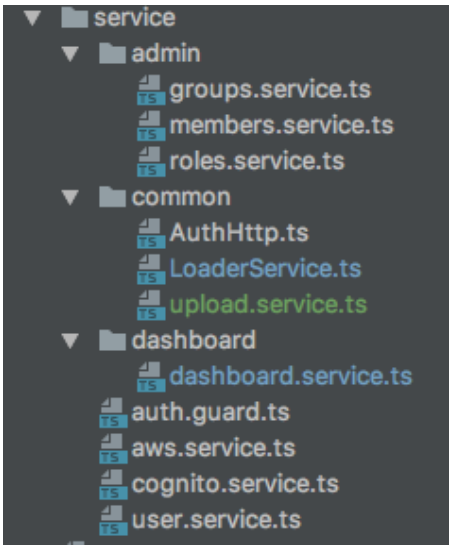

Figura 59: Servicios implementados en Jupiter

A continuación una descripción de cada uno de ellos, y con que *endpoint* se comunican, en el caso que sea necesario especificarlo:

- **auth.guard.ts**: En este archivo se definen todas las *guardas* de las rutas, es decir, interceptores de rutas que permiten securizar las mismas en función del grupo al que pertenezca el usuario, o si está autenticado
  - **AdminGuard**: Comprueba si el usuario esta autenticado y pertenece al grupo de ADMIN, en cualquier otro caso, se redirige a inicio
- **cognito.service.ts**: En este archivo se definen todos los servicios relacionados con la autenticación en Amazon Cognito, y está intimamente relacionado con user.service.ts
  - **UserRegistrationService**: Servicio encargado de registrar un usuario en AWS Cognito.
  - **UserLoginService**: Servicio encargado de realizar el Login de un usuario, una vez autenticado, se comunica con UserService, para comprobar las autorizaciones de dicho usuario.
  - **UserParametersService**: Parámetros de AWS Cognito pertenecientes al usuario logueado.
- **user.service.ts**: En este archivo se definen todos los servicios relacionados con la autorización del usuario en Base de Datos.
  - **UserService**: Servicio encargado de obtener la autorización del usuario, de comprobar si hay algún usuario logueado y de comprobar a que grupo pertenece.
- **groups.service.ts**: En este archivo se definen todos los servicios relacionados con la gestión de los grupos de la aplicación.
  - **GroupsService**: Servicio encargado de obtener los grupos, añadir y eliminar roles a un grupo, y obtención de grupos en función de rol y/o usuario.
- **members.service.ts**: En este archivo se definen todos los servicios relacionados con la gestión de los usuarios o miembros de la aplicación.
  - **MembersService**: Servicio encargado de obtener los usuarios, y añadir o eliminar usuarios.
- **roles.service.ts**: En este archivo se definen todos los servicios relacionados con la gestión de los roles de la aplicación.
  - **RolesService**: Servicio encargado de obtener los roles, y añadir o eliminar los mismos.
- **AuthHttp.ts**: En este archivo se define un *wrapper* sobre los servicios HTTP de Angular 4, para insertar dentro de cada petición un Bearer de Autorización.
- **LoaderService.ts**: Se define un servicio, que permite mostrar un *overlay* con un *loader spinner*.

- **upload.service.ts**: Contiene un servicio para permitir subir archivos a Amazon S3.
- **dashboard.service.ts**: En este archivo se definen todos los servicios relacionados con la administración de modelos.
  - **ContainerService**: Servicio encargado de la gestión de los tipos de contenedores existentes disponibles en la aplicación.
  - **InputTypeService**: Servicio encargado de la gestión de los tipos de entrada que se pueden usar para entrenar los modelos.
  - **TrainedModelService**: Incluye toda la lógica necesaria para la gestión de los modelos de la aplicación, desde su creación, hasta el arranque, parada, eliminación o testeo de los mismos.

6.3.4. Interfaz de Gestión de Usuarios

En este apartado, se mostrará la interfaz de la parte de Administración, es decir, del modelo de Gestión de Usuarios, Grupos y Roles, y se explicarán cada uno de las funcionalidades que se han implementado.

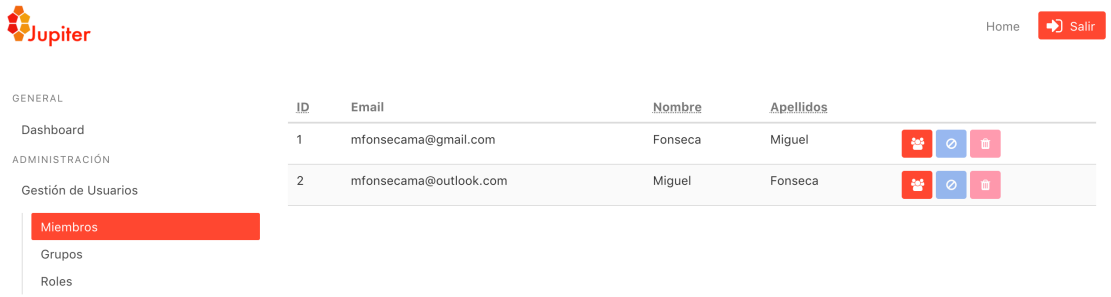

Figura 60: Listado de Miembros de Jupiter

En primer lugar, se explica la implementación de la gestión de los Usuarios de Jupiter, en la Figura 60, se observa un listado de usuarios, y una serie de opciones, de las cuales, únicamente está activa, la primera de ellas, que se trata de la asignación de grupo para el usuario.

Al hacer *click* en dicha opción, aparece la pantalla de asignación de grupo (Figura 61), en la cual, seleccionando un grupo se asigna dicho usuario a el grupo seleccionado.

Figura 61: Asignación de grupo para un usuario

Figura 62: Listado de Grupos de Jupiter

Se continuará con la explicación de la gestión de los grupos en Jupiter (Figura 62). Actualmente, no se permite la creación ni edición de grupos, ya que realmente, el diseño de la aplicación está preparado para trabajar a nivel de rol y usuario, y los grupos son meros agrupadores, sin embargo, es una funcionalidad que está contemplada en futuras modificaciones de la misma.

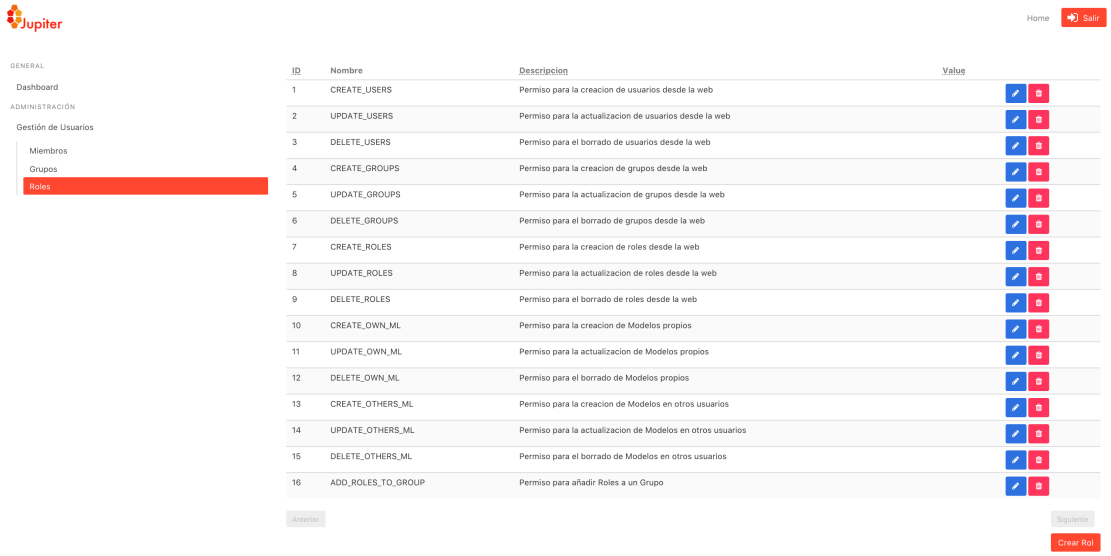

Figura 63: Listado de Roles de Jupiter

Figura 64: Creación o edición de Rol de Jupiter

Por último, la gestión de los roles, en la Figura 63, se observa el listado de todos los roles disponibles en la aplicación, y cada uno de ellos tiene dos opciones, la primera de ella, llevaría a la pantalla de edición (Figura 64), donde se puede editar el rol, asignarle nuevos grupos o quitarlo de dichos grupos, y la segunda de ellas, sería la opción de borrar rol, mediante la cual, previa confirmación (Figura 65), se borraría dicho rol y se desasignaría de los grupos.

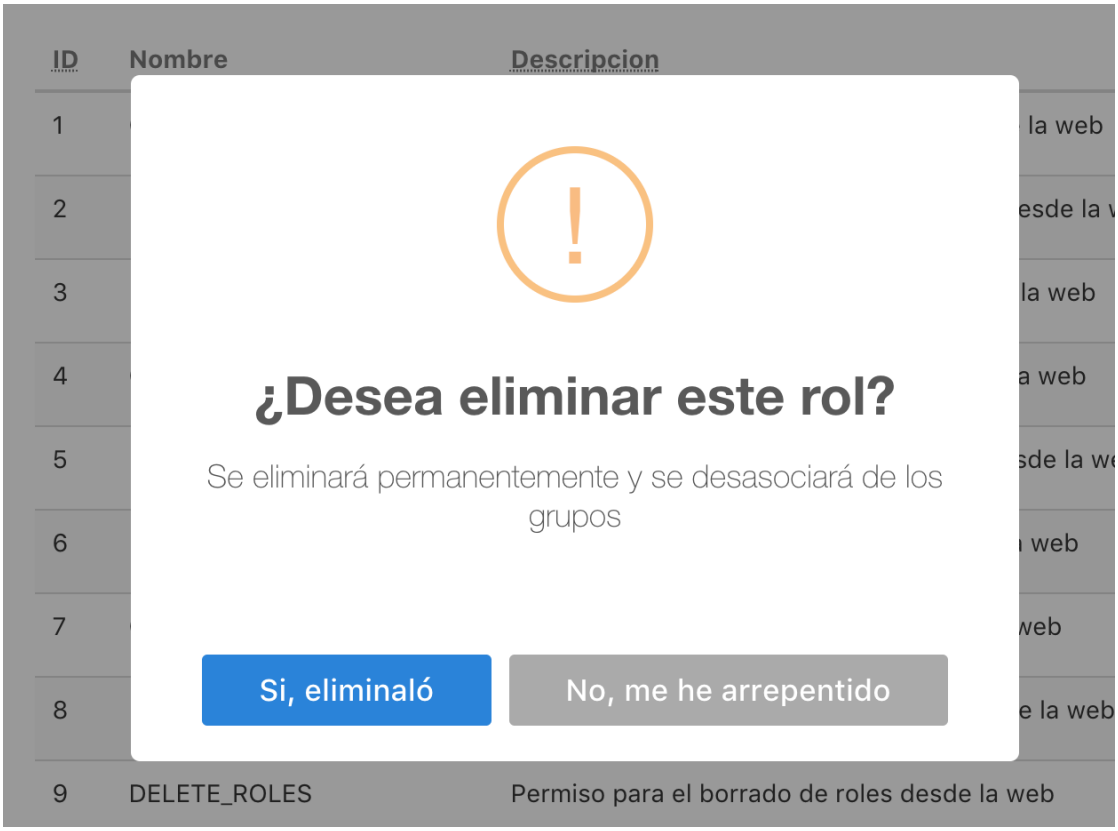

Figura 65: Confirmación de borrado de Rol

6.3.5. Interfaz de Gestión de Modelos

En este apartado, se explicará la implementación de la parte de Gestión de Modelos del Framework, es decir, el lugar donde puedes crear, arrancar, parar y probar los modelos disponibles, para que puedan usados por el API móvil que se ofrece.

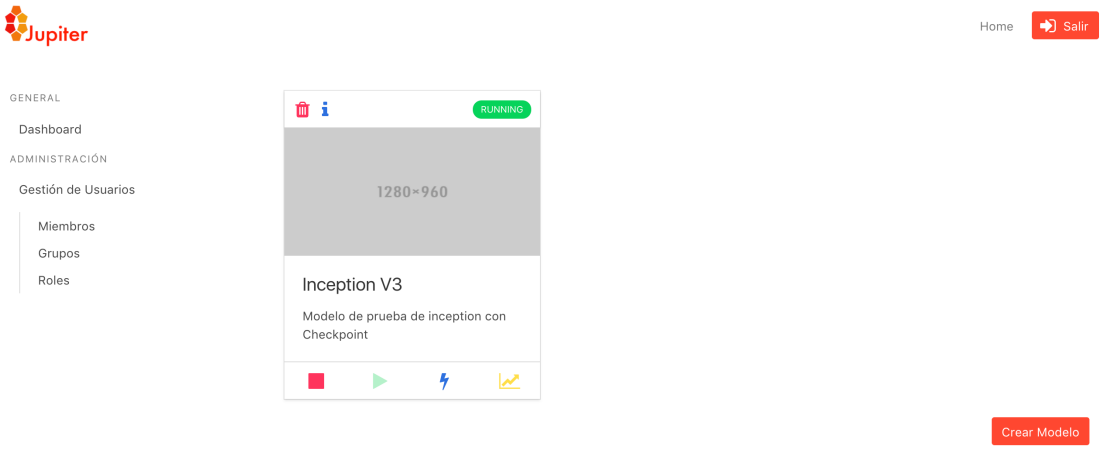

Figura 66: Dashboard de modelos del usuario en Jupiter

Nada mas autenticarse en la aplicación, se llega al Dashboard (Figura 66), donde aparecen todos los modelos que el usuario tiene creados.

Figura 67: Datos de modelo Jupiter

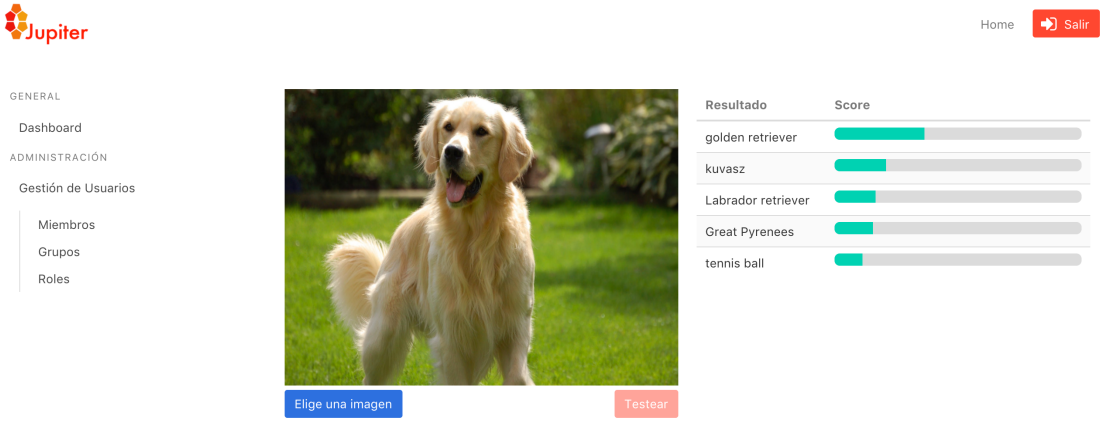

Figura 68: Testeo de modelo Jupiter

Cada uno de estos modelos, tiene diversas opciones:

- La papelera sirve para eliminar el modelo, tanto de BBDD como físicamente de S3, sin posibilidad de recuperarlo.
- El icono de información tiene como objetivo mostrar los datos del modelo, así como de su Api Key, que permite conectarse con dicho modelo desde el API móvil
- El icono de STOP permite parar el modelo
- El icono de PLAY permite arrancar el modelo

- El icono de RAYO permite probar el modelo
- El icono de STATS permite mostrar las estadísticas del modelo (Fase 2)

Por último, arriba a derecha de cada modelo, indica el estado en el cual se encuentra el modelo actualmente (Arrancando, Arrancado, Parando, Parado, Error, Preparado).

Cuando se selecciona la opción de testeo, se navega hacia la Figura 68, donde se selecciona una entrada (en este caso, una imagen, que es lo que se implementa en la Fase 1), y proporciona una salida, que se trata del etiqueta y score de dicha entrada.

6.4. Servidor WEB

Esta sección explicará como se ha implementado la parte de Backend del Framework, desde las librerias que utiliza y la conexión con la BBDD, hasta que servicios Rest están publicados y como conectarse con el Predictor desde el API móvil, o como se gestionaría la inserción de modelos en la cola SQS para la Fase 2 de la implementación.

6.4.1. Librerias

El servidor WEB está implementado con JAVA 8, y como ya se ha comentado anteriormente, utiliza Spring (En concreto, Spring Boot) como Framework. Y se usa Maven 3.5.5 como gestor de dependencias.

Más en concreto, se utilizan las siguientes librerías:

- Spring Boot: Se trata de un *wrapper* de Spring, que permite una desarrollo y una ejecución más ágil. Además, en concreto, se utilizan los siguientes módulos de Spring Boot:
  - JPA: Permite una implementación sencilla de la capa de acceso a datos.
  - Data Rest: Es una librería que permite publica servicios REST, directamente accediendo a la capa de acceso a Datos de manera automática, utilizando lo que se denomina *hypermedia-driven REST* (cita)
  - Websocket: Utiliza la tecnología de WebSockets para realizara un túnel entre servidor y cliente, y permitir intercambio de información dinámica.
- MySQL Connector: Driver de MySQL para Java
- AWS SKD Java: El SDK (Software Development Kit) de Amazon AWS, que incluye todas las librerías necesarias para conectarse con el mismo.
- Docker Java: Una librería que permite realizar operaciones de Docker desde código en JAVA
- Apache Commons Compress: Una librería de Apache para trabajar con archivos comprimidos
- GRPC: Es un Framework para conectarse entre diferentes lenguajes, en el caso que nos ocupa, se encargará de conectarse con el modelo que predice los datos.
- Spring JMS: Librería de Spring que actúa como *wrapper* del JMS de Java.
- Amazon SQS Java Messaging: Librería para interactuar con el API SQS de Amazon.

6.4.2. Compilación

Las compilaciones con Maven son simples, ya que con una configuración mínima, y apenas algún *plugin*, te permite compilar y empaquetar los proyectos de manera eficiente.

```
<build>
    <extensions>
        <extension>
            <groupId>kr.motd.maven</groupId>
            <artifactId>os-maven-plugin</artifactId>
            <version>1.4.1.Final</version>
        </extension>
    </extensions>
    <plugins>
        <plugin>
            <groupId>org.xolstice.maven.plugins</groupId>
            <artifactId>protobuf-maven-plugin</artifactId>
            <version>0.5.0</version>
            <configuration>
                <!--
                  The version of protoc must match protobuf-java. If you don't depend on
                  protobuf-java directly, you will be transitively depending on the
                  protobuf-java version that grpc depends on.
                -->
                <protocArtifact>com.google.protobuf:protoc:3.0.2:exe:${os.detected.classifier}</protocArtifact>
                <pluginId>grpc-java</pluginId>
                <pluginArtifact>io.grpc:protoc-gen-grpc-java:${grpc.version}:exe:${os.detected.classifier}</pluginArtifact>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>compile</goal>
                        <goal>compile-custom</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>

        <plugin>
            <artifactId>maven-war-plugin</artifactId>
            <configuration>
                <webResources>
                    <resource>
                        <directory>src/main/resources/.ebextensions</directory>
                        <targetPath>.ebextensions</targetPath>
                        <filtering>true</filtering>
                    </resource>
                </webResources>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Figura 69: Plugins usados en la compilación Maven

En este proyecto, cabe destacar los siguientes dos plugins:

- protobuf-maven-plugin: Este plugin permite compilar las entidades *.proto* de GRPC.io, y transformarlas en clases JAVA, de manera que mediante una implementación genérica de una entidad, te puedes conectar con otro *.proto* que este transformado a cualquier otro lenguaje de programación.
- maven-war-plugin: Se encarga de añadir lo que se indice en el directorio al WAR que se desplegará en Amazon EBS

6.4.3. Configuraciones

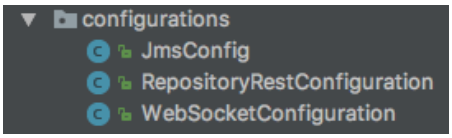

Figura 70: Configuración Específica Web Jupiter

En este apartado, nos centraremos en una serie de configuraciones que han sido necesarias para el correcto funcionamiento del servidor Web, ya que sin ellas, no hubiera sido posible implementar ciertas funcionalidades.

En primer lugar, la clase *JmsConfig* se encarga de realiza la conexión con la cola definida en Amazon SQS, permitiendo instanciar un *Bean* que contiene la capacidad de encolar mensajes en dicha cola.

A continuación, como se observa en la Figura 70, se encuentra *RepositoryRestConfiguration*, clase JAVA encargada de configurar el CORS y que se publiquen los ID's de las entidades cuando se utiliza Data Rest.

Por último, la configuración de los WebSockets en la clase *WebSocketConfiguration*

6.4.4. Punto de arranque del Servidor

```java
package com.mfonseca.jupiter;

import ...

@SpringBootApplication
@Import(RepositoryRestConfiguration.class)
@EnableAsync
@EnableScheduling
@EnableWebMvc
public class JupiterPredictApplication extends AsyncConfigurerSupport {

    public static void main(String[] args) { SpringApplication.run(JupiterPredictApplication.class, args); }

    @Override
    public Executor getAsyncExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        executor.setCorePoolSize(2);
        executor.setMaxPoolSize(2);
        executor.setQueueCapacity(500);
        executor.setThreadNamePrefix("generationContainer-");
        executor.initialize();
        return executor;
    }

    @Bean
    public FilterRegistrationBean cacheControlFilter() {
        FilterRegistrationBean registration = new FilterRegistrationBean();
        Filter filter = new CacheControlFilter();
        registration.setFilter(filter);
        registration.addUrlPatterns("/*");
        return registration;
    }

}
```

Figura 71: Arranque del Servidor Jupiter

Todos los servidores deben tener un punto de arranque del mismo. En los servidores tradicionales, este seria el *web.xml*, en el caso de Spring Boot, es una clase JAVA *JupiterPredictApplication*, y mediante la anotación @SpringBootApplication, realiza una encapsulación de la misma y se ejecuta con su propio Tomcat embebido.

En la Figura 71 se observa la clase dicha, pudiendo destacar las siguientes anotaciones y elementos:

- @EnableAsync: Permite habilitar la ejecución de código asíncrono
- @EnableScheduling: Permite habilitar la ejecución de planificadores, de manera que se ejecutan tareas cada un tiempo definido en la propia tarea.
- @EnableWebMVC: Permite la utilización de los servicios REST
- getAsyncExecutor: Define el ejecutor asíncrono

6.4.5. Entidades de Base de Datos

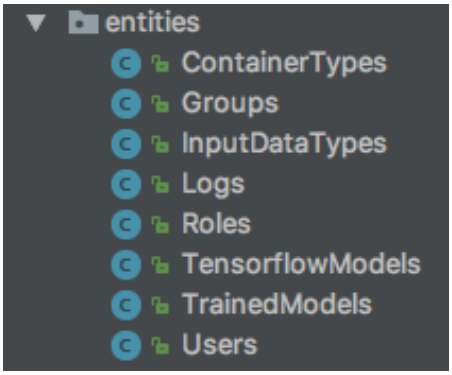

Figura 72: Entidades mapeadas Jupiter

Las entidades de Base de Datos permite establecer modelos ORM, de manera que eleva la abstracción de las columnas de las tablas de la Base de Datos a objetos JAVA, permitiendo así trabajar con una filosofía orientada a Objetos.

De manera concreta, es la representación en JAVA de las tablas de la Base de Datos.

6.4.6. Repositorios

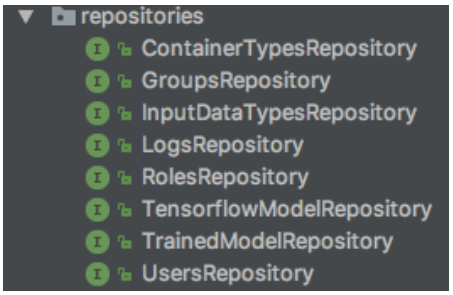

Figura 73: Repositorios Jupiter

Los repositorios actúan de puente entre los servicios, es decir, la capa de lógica de negocio, con el acceso a la Base de Datos y la realización de tareas sobre ella.

Cabe destacar, que gracias a la utilización de Spring Data Rest, estos repositorios se abstraen un nivel mas y también se publican como endpoints de acceso, de manera que mediante peticiones REST, se puede acceder a la entidad de Base de Datos con la cual esté relacionado dicho repositorio.

6.4.7. Servicios

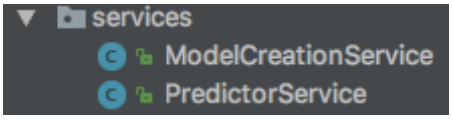

Figura 74: Servicios Jupiter

Los servicios son los encargados de realizar la lógica de negocio, es decir, tratar los inputs de los servicios y realizar lo necesario para que esos datos sean correctamente procesados o tratados.

El servidor Web, unicamente tiene dos servicios propiamente dichos, pero son pieza clave en el devenir del Framework.

ModelCreationService Este primer servicio se encarga de gestionar todo lo relacionado con la creación y eliminación de modelos, asi como de crear las tareas en el Container Registry de Amazon, y además, permite ejecutar o parar dichas tareas.

PredictorService Como su propio nombre indica, es el encargado de realizar de puente entre el Servidor Web y el Servidor GRPC que se está ejecutándose en el contenedor correspondiente al modelo sobre el cual se desee realizar la predicción.

A continuación se muestra el código relacionado con la predicción, ya que es de especial interés.

```
1
```

```
2  public ResponseDto predict(Integer trainedModelId, String
       host, int port, String modelName, InputStream stream)
       throws InterruptedException, IOException {
3
4   ResponseDto respuesta = new ResponseDto();
5
6   try {
7    createServerInstance(host, port);
8    ByteString imageData = ByteString.readFrom(stream);
9    TensorProto.Builder tensorProtoBuilder = TensorProto.
       newBuilder();
10   TensorShapeProto.Dim dim = TensorShapeProto.Dim.newBuilder
       ().setSize(1).build();
11
12   TensorShapeProto featuresShape = TensorShapeProto.
       newBuilder()
13    .addDim(dim).build();
14
15   tensorProtoBuilder.addStringVal(imageData);
16   tensorProtoBuilder.setDtype(DataType.DT_STRING);
17   tensorProtoBuilder.setTensorShape(featuresShape);
18
19   TensorProto featuresTensorProto = tensorProtoBuilder.build
       ();
20
21   Model.ModelSpec modelSpec = Model.ModelSpec.newBuilder()
22    .setName(modelName).build();
23
24   Predict.PredictRequest request = Predict.PredictRequest.
       newBuilder()
25    .setModelSpec(modelSpec)
26    .putInputs("images", featuresTensorProto)
27    .build();
28
29   try {
30    respuesta.insertPrediction(blockingStub.withDeadlineAfter
       (10,                TimeUnit.SECONDS).predict(request)
       );
31   } catch (StatusRuntimeException e) {
32     respuesta.setError(e.getStatus().getCode()+": "+e.
         getStatus().getCause().getMessage());
33     logger.warn("RPC failed: "+e.getStatus());
34   }
35
36  } finally {
37   if (respuesta.hasPrediction()) {
38    Logs logs = new Logs();
39    logs.setTrainedModelId(trainedModelId);
40    logs.setCreatedAt(new Date());
41    logs.setDevice(null);
```

```
42      logs.setFirstResult((String) respuesta.getNResult(1).
           keySet().toArray()[0]);
43      logs.setScoreFirstResult(((Float) respuesta.getNResult(1)
           .entrySet().toArray()[0]).doubleValue());
44      logs.setSecondResult((String) respuesta.getNResult(2).
           keySet().toArray()[0]);
45      logs.setScoreSecondResult(((Float) respuesta.getNResult
           (2).entrySet().toArray()[0]).doubleValue());
46      logs.setThirdResult((String) respuesta.getNResult(3).
           keySet().toArray()[0]);
47      logs.setScoreThirdResult(((Float) respuesta.getNResult(3)
           .entrySet().toArray()[0]).doubleValue());
48      logs.setFourthResult((String) respuesta.getNResult(4).
           keySet().toArray()[0]);
49      logs.setScoreFourthResult(((Float) respuesta.getNResult
           (4).entrySet().toArray()[0]).doubleValue());
50      logs.setFifthResult((String) respuesta.getNResult(5).
           keySet().toArray()[0]);
51      logs.setScoreFifthResult(((Float) respuesta.getNResult(5)
           .entrySet().toArray()[0]).doubleValue());
52      logsRepository.save(logs);
53    }
54    shutdown();
55   }
56  return respuesta;
57  }
```

6.4.8. Controladores REST

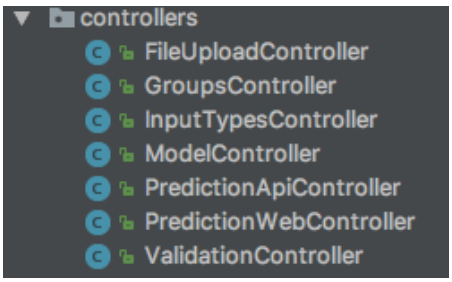

Figura 75: Controladores Jupiter

Los controladores son la capa del Servidor WEB que se encarga de realizar de puente entre la aplicación WEB o los clientes que consumen el Servidor, y la capa de servicios.

Cabe señalar, que se trata de controladores REST, es decir, son controladores que responden peticiones REST, por lo que devuelve el estado de una aplicación (JSON con los datos) y no la vista o una referencia a una vista que sería compilada en el servidor.

Entre todos los controladores REST, destacan los tres siguientes

ModelController

Es el controlador asociado a la creación de modelos, arrancar, parar, borrar y obtener el API Key de dicho modelo.

PredictionWebController

Es el controlador encargado de atender las peticiones de predicción desde la aplicación WEB (La opción de testeo de modelo), ya que la manera de tratar la seguridad de la aplicación es diferente desde la WEB que desde el API.

PredictionApiController

Es el controlador encargado de atender las peticiones de predicción desde el API móvil, por lo mismo que se indica en el apartado anterior, está separado debido a que la seguridad se trata de manera completamente diferente.

6.4.9. API Móvil

En esta sección se explicarán los dos API's implementados, muy simples, tanto para el lenguaje JAVA (Android) como para Swift 3 (iOS)

JAVA (Android)

```
<groupId>com.mfonsecama.jupiter</groupId>
<artifactId>jupiter-java</artifactId>
<version>1.0.0.0</version>
```

Figura 76: API Java Android

Para utilizar este API en java se ha de importar la librería de la Figura 76, y con una implementación simple, resumida en las siguientes lineas, se puede utilizar de manera intuitiva:

```
1
2  JupiterApi api = JupiterApiClient.builder()
3    .withCredentials(new PropertiesVariables())
4    .build();
5  File is = new File("/Users/mfonsecama/Desktop/martillo.jpg")
       ;
6  JupiterPredictRequest jupiterPredictRequest = new
       JupiterPredictRequest(modelKey, is);
7
8  JupiterPredictResponse response = api.predict(
       jupiterPredictRequest);
```

Si se observa la implementación anterior, en las properties se definirá la seguridad de la aplicación, en concreto buscará:

- **JUPITER_AUTHORIZATION_KEY** = *xxxx* donde *xxxx* es el nombre de usuario en Base64 con el cual se ha registrado en la aplicación.

Swift 3 (iOS)

Se trata del análogo de la librería en Android, pero implementada en Swift 3, para que pueda ser utilizada en dispositivos iOS.

Es igual de simple que el anterior, siendo un ejemplo de implementación el siguiente:

```
1
2  func testExample() {
3          // This is an example of a functional test case.
4          // Use XCTAssert and related functions to verify
              your tests produce the correct results.
5
6          let expectat = expectation(description: "SomeService
               does stuff and runs the callback closure")
7
8          if let path = self.bundle!.url(forResource: "
              martillo", withExtension: "jpg") {
9                let jupiterPredictRequest =
                    JupiterPredictRequest(modelApiKeyCons: "
                    yyyyy", imageUrlCons: path)
10               let jupiterApi = JupiterApiClient.builder().
                    addCredentials(credentials:
                    ResourcesPropertiesCredentials(jupiterKey
                    : "xxxxx" )).build()
11               jupiterApi.predict(request:
                    jupiterPredictRequest) { (result:
                    JupiterPredictResponse) in
12                       expectat.fulfill()
13               }
14         }
15
16         waitForExpectations(timeout: 20) { error in
17                if let error = error {
18                       XCTFail("
                            waitForExpectationsWithTimeout
                            errored: \(error)")
19                }
20         }
21 }
```

Si se observa la implementación anterior, la credenciales se crean a partir de ResourcesPropertiesCredentials(jupiterKey: yyyyy) donde *yyyyy* es el nombre de usuario en Base64 con el cual se ha registrado en la aplicación.

6.4.10. Colas SQS

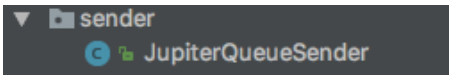

Figura 77: Colas SQS Jupiter

La clase *JupiterQueueSender* es la encargada de encolar cualquier modelo que esté pendiente de entrenar y serán los **MIE's** los que desencolaran dichos modelos para proceder al entrenamiento.

6.5. Módulo Integrado de Entrenamiento (MIE)

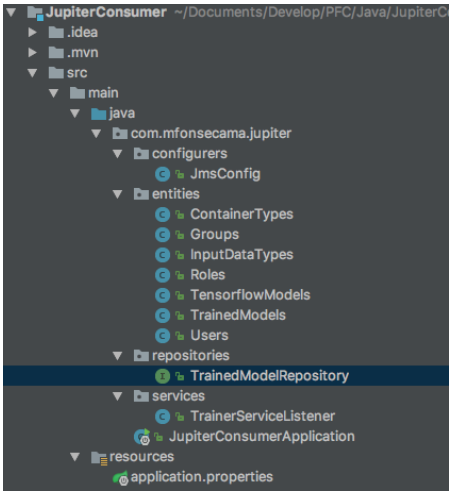

Figura 78: Estructura del Módulo Integrado de Entrenamiento (MIE)

Los Módulos Integrados de Entrenamiento son aplicaciones que se ejecutan en cada servidor capaz de realizar un entrenamiento, y tiene como objetivo, desencolar peticiones de entrenamiento y comenzar dicho entrenamiento, y cada cierto tiempo, ir actualizando el progreso del entrenamiento y almacenando los checkpoints en Amazon S3.

Es una implementación que esta englobada dentro de la Fase 2 del proyecto, y que por tanto, no está desarrollada en esta primera fase, sin embargo, es importante conocer que es un módulo que es clave para completar el ciclo de entrenamiento completo.

7. Validación y pruebas

7.1. Validación

Con la implementación propuesta se validan los siguientes requisitos funcionales y no funcionales de la Fase 1.

Identificador	Descripción	Validado
RF1ID1	Creación de Roles, Grupos y Usuarios predefinidos	Sí
RF1ID2	El usuario Administrador, perteneciente al grupo ADMIN, puede crear roles	Sí
RF1ID3	El usuario Administrador, perteneciente al grupo ADMIN, puede crear grupos	Sí
RF1ID4	El usuario Administrador, perteneciente al grupo ADMIN, puede dar de alta usuarios	Sí
RF1ID5	El usuario Administrador, perteneciente al grupo ADMIN, puede asignar usuarios a grupos	Sí
RF1ID6	El usuario Administrador, perteneciente al grupo ADMIN, puede asignar roles a grupos	Sí
RF1ID7	Un usuario puede darse de alta y se le asigna el grupo JUNO	Sí
RF1ID8	Creación de modelos predefinidos y habilitar los mismos	Sí
RF1ID9	Usuarios con permisos pueden crear un modelo pre-definido usando su propio modelo pre-entrenado	Sí
RF1ID10	Usuarios con permisos pueden arrancar sus modelos pre-definidos	Sí
RF1ID11	Usuarios con permisos pueden parar sus modelos pre-definidos	Sí
RF1ID12	Usuarios con permisos pueden probar sus modelos pre-definidos	Sí
RF1ID13	Usuarios con permisos pueden eliminar sus modelos pre-definidos	Sí
RF1ID14	Desarrollador puede conectarse con su modelo mediante API Java	Sí
RF1ID15	Desarrollador puede conectarse con su modelo mediante API Swift	Sí
RNFD1	La aplicación WEB debe funcionar en todos los navegadores	Sí
RNFD2	Los usuarios pueden contactar con el soporte del Framework para proponer modelos	No
RNFD3	El sistema debe incorporar un API CLI para testeo desde linea de comandos	No
RNFD4	El framework debe ser tolerante a fallos	Sí

Tabla 4: Requisitos Funcionales y no Funcionales validados de la Fase 1

7.2. Pruebas

Para la realización de las pruebas se han implementado test unitarios tanto en JUnit como en XCode, para evaluar el API implementado de cara al desarrollador.

Por otro lado, en el servidor, se han implementado test unitarios de los servicios REST, así como del API de predicción.

8. Planificación y Presupuesto

8.1. Planificación

Se ha planificado el proyecto en función de las necesidades de análisis y desarrollo del mismo, y se ha estructurado por requisito funcional. Los requisitos no funcionales no se incluyen en la planificación debido a que son opcionales y solo serán implementados en el caso de que el desarrollo se encuentre dentro de las fechas estimadas.

#	Estado	Asunto	Autor	Tipo	Fecha de inicio	Tiempo estimado	Tiempo dedicado
11928	Nueva	RF2ID7	Miguel Fonseca	Nueva Funcionalidad	01/09/2017	150.0	0
11927	Nueva	RF2ID6	Miguel Fonseca	Nueva Funcionalidad	17/08/2017	120.0	0
11926	Nueva	RF2ID5	Miguel Fonseca	Nueva Funcionalidad	29/07/2017	100.0	0
11925	Nueva	RF2ID4	Miguel Fonseca	Nueva Funcionalidad	18/07/2017	65.0	0
11924	Nueva	RF2ID3	Miguel Fonseca	Nueva Funcionalidad	04/07/2017	100.0	0
11923	En curso	RF2ID2	Miguel Fonseca	Nueva Funcionalidad	25/06/2017	40.0	0
11922	Resuelta	RF2ID1	Miguel Fonseca	Nueva Funcionalidad	21/06/2017	20.0	16.0
11920	Resuelta	RF1ID15	Miguel Fonseca	Nueva Funcionalidad	14/06/2017	30.0	33.0
11919	Resuelta	RF1ID14	Miguel Fonseca	Nueva Funcionalidad	10/06/2017	30.0	21.0
11918	Resuelta	RF1ID13	Miguel Fonseca	Nueva Funcionalidad	07/06/2017	15.0	11.0
11917	Resuelta	RF1ID12	Miguel Fonseca	Nueva Funcionalidad	02/06/2017	30.0	12.0
11916	Resuelta	RF1ID11	Miguel Fonseca	Nueva Funcionalidad	31/05/2017	20.0	14.0
11915	Resuelta	RF1ID10	Miguel Fonseca	Nueva Funcionalidad	25/05/2017	30.0	22.0
11914	Resuelta	RF1ID9	Miguel Fonseca	Nueva Funcionalidad	12/05/2017	100.0	66.0
11913	Resuelta	RF1ID8	Miguel Fonseca	Nueva Funcionalidad	28/04/2017	100.0	75.0
11912	Resuelta	RF1ID7	Miguel Fonseca	Nueva Funcionalidad	19/04/2017	60.0	43.0
11911	Resuelta	RF1ID6	Miguel Fonseca	Nueva Funcionalidad	16/04/2017	10.0	6.0
11910	Resuelta	RF1ID5	Miguel Fonseca	Nueva Funcionalidad	15/04/2017	10.0	17.0
11908	Resuelta	RF1ID4	Miguel Fonseca	Nueva Funcionalidad	14/04/2017	10.0	8.0
11907	Resuelta	RF1ID3	Miguel Fonseca	Nueva Funcionalidad	12/04/2017	10.0	6.0
11906	Resuelta	RF1ID2	Miguel Fonseca	Nueva Funcionalidad	09/04/2017	10.0	7.0
11905	Resuelta	RF1ID1	Miguel Fonseca	Nueva Funcionalidad	05/04/2017	20.0	14.0

Figura 79: Planificación de tareas según identificador

En la figura 79 se muestra la lista de tareas, con sus fechas estimadas y sus tiempos estimados y realizados. En la figura 80 se muestra el diagrama de Gantt correspondiente con la planificación de la figura 79.

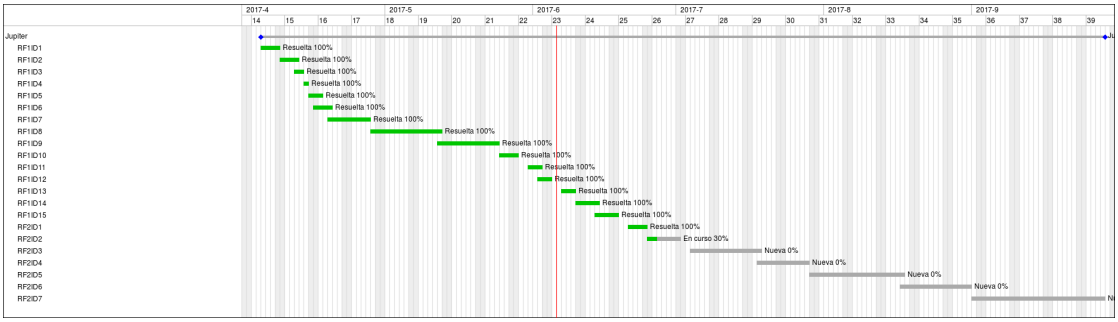

Figura 80: Diagrama Gantt de las tareas de Jupiter

Para la realización de estas estimaciones, se ha realizado el software de gestión de proyectos *Redmine*

8.2. Presupuesto

Se divide el presupuesto en tres categorías para facilitar la comprensión del mismo, y finalmente, se realizará un presupuesto completo agrupando estas tres categorías.

8.2.1. Costes de Software y Material

El coste de software y material se establece mediante el coste de adquisición de los mismos. La tabla 5 lista el software y el material adquirido:

Descripción	**Cant.**	**Coste (€)**	**Amort.**	**Uso**	**Coste Imputable (€)**
MacBook Pro 13'	1	2400	60 meses	6 meses	240
Licencia IntelliJ IDEA	1	49.90	-	6 meses	299.4
TOTAL					**539.4**

Tabla 5: Costes de Software y Material

8.2.2. Costes de Personal

El coste de personal se establece mediante el calculo del coste anual del desarrollador y el número de horas trabajadas (o estimadas en el caso de requisitos no desarrollados).

El coste anual de un desarrollador *full-stack* senior es, de aproximadamente, 38€ por hora.

La tabla 6 lista dichos costes.

Desarrollador	**Salario Hora (€)**	**Horas totales**	**Coste (€)**
Miguel Fonseca	38	956 horas	36328
TOTAL			**36328**

Tabla 6: Costes de Personal

8.2.3. Costes Indirectos

Los costes indirectos representan los costes de mantenimiento de servidores y de toda la infraestructura necesaria para el correcto funcionamiento del Framework. Solo se estimarán los costes del tiempo que dure el desarrollo.

La tabla 7 muestra dichos costes.

Descripción	**Coste / mes (€)**	Meses	Coste (€)
Instancia EC2 para Contenedores Docker	14.64	6 meses	87.84
Instancia EC2 para ejecución servidor Web	14.64	6 meses	87.84
Amazon S3 (100 GB)	2.30	6 meses	13.8
Amazon RDS (BBDD)	26.26	6 meses	157.56
Amazon SQS (1000000 peticiones)	0	6 meses	0
TOTAL			**347.04**

Tabla 7: Costes Indirectos

8.2.4. Coste total

Para terminar, el coste total de los 6 meses de desarrollo sería el indicado en la tabla 8

Descripción	Coste (€)
Costes de Software y Material	539.4
Costes de Personal	36328
Costes Indirectos	347.04
TOTAL	**37214.44**

Tabla 8: Costes Totales

9. Marco Regulador

Para el desarrollo correcto del Proyecto, es necesario cumplir una serie de legislaciones y normativas vigentes, tanto a nivel español como a nivel europeo.

9.1. Ley de Protección de Datos

Al tratarse de un Proyecto en el cual se almacenan datos sensible de los usuarios y clientes, desde nombre y apellidos hasta datos bancarios, es necesario cumplir esta Ley.

9.1.1. Marco español

La Ley Orgánica 15/1999, 13 de diciembre, de Protección de Datos de Carácter Personal (LOPD)

Esta Ley Orgánica "tiene por objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor e intimidad personal y familiar."

Por lo tanto, en cualquier caso en el que cualquier proyecto se encuentre en esta disyunción, ha de aplicar la **LOPD**

9.1.2. Marco europeo

Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo

El 27 de Abril del 2016 se aprobó el Reglamento de Protección de Datos de la Unión Europea. Tiene como aspectos claves, lo siguientes:

- Derecho al olvido: Es la consecuencia del derecho que tiene los ciudadanos a que los datos personales sean suprimidos en el caso de que ya no sean necesarios o cuando hayan sido obtenidos de manera ilícita.
- Derecho de Portabilidad: El interesado que proporciona unos datos que se estén tratando de manera automatica puede solicitar esos datos de manera que puedan ser trasladados a otro responsable.

Al aprobarse en 2016, esta Ley entrará en vigor en el año 2018

9.2. Ley de Servicios de de la Sociedad de de la Información y del Comercio Electrónico

Esta Ley obliga al sitio web a proporcionar información sobre la empresa, así como de los registros administrativos que contenga.

9.2.1. Marco español

Ley 34/2002, de 11 de julio, de servicios de la sociedad de la información y de comercio electrónico

La presente Ley propone "la regulación del régimen jurídico de los servicios de la sociedad de la información y de la contratación por vía electrónica, en lo referente a las obligaciones de los prestadores de servicios incluidos los que actúan como intermediarios en la transmisión de contenidos por las redes de telecomunicaciones, las comunicaciones comerciales por vía electrónica, la información previa y posterior a la celebración de contratos electrónicos, las condiciones relativas a su validez y eficacia y el régimen sancionador aplicable a los prestadores de servicios de la sociedad de la información."

9.2.2. Marco europeo

Directiva 2000/31/CE del Parlamento Europeo y del Consejo, de 8 de junio de 2000, relativa a determinados aspectos jurídicos de los servicios de la sociedad de la información, en particular el comercio electrónico en el mercado interior

No se trata exactamente de la normativa europea análoga a la indicada dentro del marco español, sin embargo si determina una serie de directivas convenientes respecto al comercio electrónico. Citando "El objetivo de la presente Directiva es contribuir al correcto funcionamiento del mercado interior garantizando la libre circulación de los servicios de la sociedad de la información entre los Estados miembros."

9.3. Condiciones Generales de la Contratación

9.3.1. Marco español

Ley 7/1998, de 13 de abril, sobre Condiciones Generales de la Contratación

Citando textualmente "La Ley tiene por objeto la transposición de la Directiva 93/13/CEE, del Consejo, de 5 de abril de 1993, sobre cláusulas abusivas en los contratos celebrados con consumidores, así como la regulación de las condiciones generales de la contratación, y se dicta en virtud de los títulos competenciales que la Constitución Española atribuye en exclusiva al Estado en el artículo 149.1.6.ª y 8.ª, por afectar a la legislación mercantil y civil."

Queda claro que el objetivo de la misma, es adaptarse a la Directiva de la unión europea en virtud a los contratos abusivos realizados con consumidores, así como la regulación de los contratos realizados por la empresa.

9.3.2. Marco europeo

Directiva 93/13/CEE del Consejo, de 5 de abril de 1993, sobre las cláusulas abusivas en los contratos celebrados con consumidores

Esta directiva de la unión europea tiene el siguiente propósito:

- Aproximar las disposiciones legales, reglamentarias y administrativas de los Estados miembros sobre las cláusulas abusivas en los contratos celebrados entre profesionales y consumidores.

Sin embargo puntualiza, que las condiciones legales o imperativas, así como aquellos llevados a cabo en convenios internacionales, no están sometidos a la Directiva

10. Entorno socio-económico

El entorno socio-económico se centra en la explotación de modelos de *Machine Learning* y en proporcionar un método de acceso seguro y sencillo a dichos datos.

Actualmente, al tratarse un prototipo de una aplicación web, o lo que se denomina en términos de desarrollo de software una *pre-alpha*, el impacto social y económico que proporciona es bastante limitado , sin embargo, en el futuro, si se consigue llegar a fases productivas, o incluso lanzar una versión candidata (o en términos anglosajones, una *release*), puede aportar una cantidad ingente de beneficios a la sociedad, y por supuesto, a su economía.

Por tanto, el objetivo final de este Proyecto, no es unicamente proporcionar un puente para que un modelo de *Machine Learning* pueda ser explotado, si no, que dichas consultas por parte de usuarios, sean auditadas y explotadas. Y de esta manera, conseguir realizar estudios de mercado sobre que intereses tienen o dejan de tener los usuarios y vender esta información a empresas que la requieran.

Es en este punto donde aparece el verdadero potencial de la aplicación, ya que sería capaz de analizar y transmitir una conclusión sobre los intereses sobre los modelos. Por ejemplo, en un modelo en el que se identifiquen marcas y modelos de coche a partir de una foto, se puede realizar mediante la auditoría de dichas fotos, que coches interesan según la zona geográfica, e incluso, a partir del dispositivo móvil con el cual realizan la fotografía. Por tanto, queda claro que las posibilidades son *quasi* ilimitadas.

Además, utiliza tecnologías basadas en la computación en la nube, eso implica que el mantenimiento de servidores y datos, se realiza por parte de empresas especializadas. Sin embargo, respecto a este último punto, se presenta un conflicto respecto al tratamiento de datos personales a nivel social, y es que no todos los usuarios querrán mandar sus fotos o sus intereses.

11. Conclusiones

En este capítulo, se presentan las conclusiones respecto a la realización del Proyecto, además de futuras ampliaciones o desarrollos, que permitan explotar al máximo los beneficios del uso del mismo.

11.1. Conclusiones

La necesidad de aprovechar al máximo la capacidad de explotación del Machine Learning fue clave a la hora de elegir, pensar y diseñar este proyecto. Con las velocidades de vértigo que está cogiendo el *Machine Learning*, era necesario una plataforma que pudiese con la mínima configuración, explotar cualquier modelo entrenado previamente, y además, tener la capacidad de poder entrenar ciertos modelos pre-definidos con algoritmos previamente diseñados.

Aprovechar las nuevas tecnologías es donde se encuentra la clave de este proyecto. Se utiliza una infraestructura completamente basada en el *cloud-computing*, donde es un proveedor el que proporciona la infraestructura sobre la cual se va a desarrollar nuestro producto. Además, nuevas tecnologías de desarrollo, como el uso de contenedores Docker, Angular 4 con apenas unos meses de vida, Spring 4, un framework muy reciente que se ejecuta sobre Java 8. Todo esto unido a permitido el diseño y realización de una plataforma dinámica de predicción.

Por supuesto, la creación de un API de acceso desde dispositivos móviles, es la otra clave del proyecto, ya que, si no se hubiesen proporcionado recursos para poder explotar dichos modelos de una manera sencilla, no se hubiese llegado a ningún lado. Está muy bien que se proporcione un framework donde poder almacenar y conectar tus modelos, sin embargo si no se va a ser capaz de explotarlos, no se va a utilizar. Éste es el objetivo del API.

Por último, y como conclusión final, que al tratarse de un (permítanme la expresión) *mundillo* que avanza tan rápido, y al ser un proyecto individual, es imposible a pesar de todo el esfuerzo del alumno, mantener todas las tecnologías al día y avanzar al mismo ritmo que avanza el mercado.

Como curiosidad, destacar que hace apenas unos días, *Apple* presentó, junto con iOS 11, su propia API de Machine Learning, denominada *Core ML*[21] que viene a presentar algo parecido a lo que realiza el API que se ha implementado con la diferencia de que el modelo se ejecuta en el dispositivo, en lugar de en un servidor externo.

11.2. Futuras mejoras e implementaciones

Existen múltiples áreas de mejora de la aplicación, así como futuras implementaciones que pueden ser interesantes.

[21] `https://developer.apple.com/documentation/coreml`

- **Conexión con los principales API's de empresas**: Permitir la posibilidad de que el usuario utilice modelos entrenados en otros frameworks, como pueda ser el de Google, pero manteniendo el API creado en Júpiter podría ser un añadido interesante.
- **La creación de una interfaz gráfica para crear modelos**: Una interfaz gráfica basada en bloques (como pueda ser *Scratch*) para crear los modelos, y que permita su descarga como código autogenerado o su utilización en futuros entrenamientos.
- **Proporcionar un API gráfico en los dispositivos**: Unos componentes gráficos, además del API programático, que permita la integración de resultados mediante gráficas, o un API de acceso al historial

12. Anexo 1: Script de Base de Datos

```
CREATE TABLE container_types
(
 container_type_id INT(11) PRIMARY KEY NOT NULL AUTO_INCREMENT,
 description LONGTEXT,
 image_url LONGTEXT,
 name VARCHAR(200) NOT NULL
);

CREATE TABLE groups
(
 group_id INT(11) PRIMARY KEY NOT NULL,
 description LONGTEXT,
 max_ml INT(11) NOT NULL,
 name VARCHAR(200) NOT NULL
);

CREATE TABLE roles
(
 roles_id INT(11) PRIMARY KEY NOT NULL,
 description VARCHAR(200),
 name VARCHAR(200) NOT NULL,
 value VARCHAR(10) NOT NULL
);

CREATE TABLE groups_roles
(
 group_id INT(11) NOT NULL,
 roles_id INT(11) NOT NULL,
 CONSTRAINT GROUP_ID_FK FOREIGN KEY (group_id) REFERENCES groups (group_id),
 CONSTRAINT ROLES_ID_FK FOREIGN KEY (roles_id) REFERENCES roles (roles_id)
);

CREATE INDEX GROUP_ID_FK ON groups_roles (group_id);
CREATE INDEX ROLES_ID_FK ON groups_roles (roles_id);

CREATE TABLE input_data_types
(
 input_data_types_id INT(11) PRIMARY KEY NOT NULL AUTO_INCREMENT,
 description LONGTEXT,
 image_url LONGTEXT,
 name VARCHAR(200) NOT NULL
);

CREATE TABLE tensorflow_models
(
```

```
 tensorflow_model_id INT(11) PRIMARY KEY NOT NULL AUTO_INCREMENT,
 cost DECIMAL(2),
 description LONGTEXT,
 docker_file_url LONGTEXT,
 enabled TINYINT(4),
 name VARCHAR(200) NOT NULL,
 container_type_id INT(11) NOT NULL,
 input_data_type_id INT(11) NOT NULL,
 CONSTRAINT CONTAINER_TYPE_ID_FK FOREIGN KEY (container_type_id) REFERENCES contain
 CONSTRAINT INPUT_TYPE_ID_FK FOREIGN KEY (input_data_type_id) REFERENCES input_data
);

CREATE INDEX CONTAINER_TYPE_ID_FK ON tensorflow_models (container_type_id);
CREATE INDEX INPUT_TYPE_ID_FK ON tensorflow_models (input_data_type_id);

CREATE TABLE users
(
 user_id INT(11) PRIMARY KEY NOT NULL AUTO_INCREMENT,
 cognito_id VARCHAR(200) NOT NULL,
 email VARCHAR(200) NOT NULL,
 given_name VARCHAR(200),
 name VARCHAR(200),
 group_id INT(11) NOT NULL,
 CONSTRAINT GROUP_ID_US_FK FOREIGN KEY (group_id) REFERENCES groups (group_id)
);

CREATE INDEX GROUP_ID_US_FK ON users (group_id);

CREATE TABLE trained_models
(
 trained_model_id INT(11) PRIMARY KEY NOT NULL AUTO_INCREMENT,
 cluster LONGTEXT,
 description LONGTEXT,
 host LONGTEXT,
 model_key LONGTEXT,
 name VARCHAR(200) NOT NULL,
 port INT(11),
 pre_trained_path LONGTEXT,
 status LONGTEXT,
 tensorflow_model_id INT(11) NOT NULL,
 user_id INT(11) NOT NULL,
 error LONGTEXT,
 CONSTRAINT TENSORFLOW_MODEL_ID_FK FOREIGN KEY (tensorflow_model_id) REFERENCES ten
 CONSTRAINT USER_ID_TF_FK FOREIGN KEY (user_id) REFERENCES users (user_id)
);

CREATE INDEX TENSORFLOW_MODEL_ID_FK ON trained_models (tensorflow_model_id);
```

```
CREATE INDEX USER_ID_TF_FK ON trained_models (user_id);

CREATE TABLE logs
(
 logs_id INT(11) PRIMARY KEY NOT NULL AUTO_INCREMENT,
 created_at DATETIME NOT NULL,
 device LONGTEXT,
 fifth_result LONGTEXT,
 first_result LONGTEXT,
 fourth_result LONGTEXT,
 latitude DOUBLE,
 longitude DOUBLE,
 score_fifth_result DOUBLE,
 score_first_result DOUBLE,
 score_fourth_result DOUBLE,
 score_second_result DOUBLE,
 score_third_result DOUBLE,
 second_result LONGTEXT,
 third_result LONGTEXT,
 trained_model_id INT(11) NOT NULL
);
```

13. Anexo 2: Datos precargados en la Base de Datos

```
INSERT INTO roles (roles_id, name, description, value)
VALUES (1, 'CREATE_USERS',
'Permiso para la creacion de usuarios desde la web', '');

INSERT INTO roles (roles_id, name, description, value)
VALUES (2, 'UPDATE_USERS',
'Permiso para la actualizacion de usuarios desde la web', '');

INSERT INTO roles (roles_id, name, description, value)
VALUES (3, 'DELETE_USERS',
'Permiso para el borrado de usuarios desde la web', '');

INSERT INTO roles (roles_id, name, description, value)
VALUES (4, 'CREATE_GROUPS',
'Permiso para la creacion de grupos desde la web', '');

INSERT INTO roles (roles_id, name, description, value)
VALUES (5, 'UPDATE_GROUPS',
'Permiso para la actualizacion de grupos desde la web', '');

INSERT INTO roles (roles_id, name, description, value)
VALUES (6, 'DELETE_GROUPS',
'Permiso para el borrado de grupos desde la web', '');

INSERT INTO roles (roles_id, name, description, value)
VALUES (7, 'CREATE_ROLES',
'Permiso para la creacion de roles desde la web', '');

INSERT INTO roles (roles_id, name, description, value)
VALUES (8, 'UPDATE_ROLES',
'Permiso para la actualizacion de roles desde la web', '');

INSERT INTO roles (roles_id, name, description, value)
VALUES (9, 'DELETE_ROLES',
'Permiso para el borrado de roles desde la web', '');

INSERT INTO roles (roles_id, name, description, value)
VALUES (10, 'CREATE_OWN_ML',
'Permiso para la creacion de Modelos propios', '');

INSERT INTO roles (roles_id, name, description, value)
VALUES (11, 'UPDATE_OWN_ML',
```

```
'Permiso para la actualizacion de Modelos propios', '');

INSERT INTO roles (roles_id, name, description, value)
VALUES (12, 'DELETE_OWN_ML',
'Permiso para el borrado de Modelos propios', '');

INSERT INTO roles (roles_id, name, description, value)
VALUES (13, 'CREATE_OTHERS_ML',
'Permiso para la creacion de Modelos en otros usuarios', '');

INSERT INTO roles (roles_id, name, description, value)
VALUES (14, 'UPDATE_OTHERS_ML',
'Permiso para la actualizacion de Modelos en otros usuarios', '');

INSERT INTO roles (roles_id, name, description, value)
VALUES (15, 'DELETE_OTHERS_ML',
'Permiso para el borrado de Modelos en otros usuarios', '');

INSERT INTO roles (roles_id, name, description, value)
VALUES (16, 'ADD_ROLES_TO_GROUP',
'Permiso para añadir Roles a un Grupo', '');

INSERT INTO groups (group_id, name, description, max_ml)
VALUES (1, 'ADMIN', 'Grupo de Administrador', '10');

INSERT INTO groups (group_id, name, description, max_ml)
VALUES (2, 'JUNO', 'Grupo de usuarios FREE', '1');

INSERT INTO groups (group_id, name, description, max_ml)
VALUES (3, 'NEPTUNO', 'Grupo de usuarios MEMBRESIA', '3');

INSERT INTO groups (group_id, name, description, max_ml)
VALUES (4, 'MARTE', 'Grupo de usuarios Pay To Use', '10');

INSERT INTO groups_roles
VALUES (1, 1);
INSERT INTO groups_roles
VALUES (1, 2);
INSERT INTO groups_roles
VALUES (1, 3);
INSERT INTO groups_roles
VALUES (1, 4);
INSERT INTO groups_roles
VALUES (1, 5);
INSERT INTO groups_roles
VALUES (1, 6);
INSERT INTO groups_roles
```

```
VALUES (1, 7);
INSERT INTO groups_roles
VALUES (1, 8);
INSERT INTO groups_roles
VALUES (1, 9);
INSERT INTO groups_roles
VALUES (1, 10);
INSERT INTO groups_roles
VALUES (1, 11);
INSERT INTO groups_roles
VALUES (1, 12);
INSERT INTO groups_roles
VALUES (1, 13);
INSERT INTO groups_roles
VALUES (1, 14);
INSERT INTO groups_roles
VALUES (1, 15);
INSERT INTO groups_roles
VALUES (1, 16);

INSERT INTO container_types (container_type_id, name, description)
VALUES (1, 'Inception V3', 'Contenedor modelo para Inception V3');

INSERT INTO input_data_types (input_data_types_id, name, description, image_url)
VALUES (1, 'Base de Datos',
 'Obtiene los datos de entrenamiento a partir de una Base de Datos',
'https://s3-eu-west-1.amazonaws.com/jupiter-static-resources/Database-300x300.png')

INSERT INTO input_data_types (input_data_types_id, name, description, image_url)
VALUES (2, 'S3',
'Obtiene los datos de entrenamiento a partir de AWS S3 (almacen de ficheros)',
'https://s3-eu-west-1.amazonaws.com/jupiter-static-resources/s3.png');

INSERT INTO input_data_types (input_data_types_id, name, description, image_url)
VALUES (3, 'Pre-trained',
'Obtiene los datos a partir de un modelo pre-entrenado',
'https://s3-eu-west-1.amazonaws.com/jupiter-static-resources/tf-logo.png');

INSERT INTO tensorflow_models (tensorflow_model_id, name,
description, cost, enabled, input_data_type_id, container_type_id)
VALUES (1, 'I-V3-BBDD', 'Inception V3 con input desde BBDD',
 0, FALSE ,1, 1);

INSERT INTO tensorflow_models (tensorflow_model_id, name,
description, cost, enabled, input_data_type_id, container_type_id)
VALUES (2, 'I-V3-S3', 'Inception V3 con input desde S3',
0, FALSE ,2, 1);
```

```
INSERT INTO tensorflow_models (tensorflow_model_id, name,
description, cost, enabled, input_data_type_id, container_type_id)
VALUES (3, 'I-V3-PT', 'Inception V3 con input desde modelo pre-entrenado',
0, TRUE ,3, 1);

INSERT INTO users VALUES (
1,'20b45291-05de-445e-9ea4-00811889edb0','mfonsecama@gmail.com',
'Fonseca','Miguel',1);
```

Referencias

[1] Jason Brownlee. *What is Deep Learning?* Ago. de 2016. URL: http://machinelearningmastery.com/what-is-deep-learning/.

[2] NVIDIA Developers. *Deep Learning Frameworks*. URL: https://developer.nvidia.com/deep-learning-frameworks.

[3] IBM. *What is cloud computing?* URL: https://www.ibm.com/cloud-computing/learn-more/what-is-cloud-computing/.

[4] Amazon AWS. *Types of Cloud Computing*. URL: https://aws.amazon.com/es/types-of-cloud-computing/.

[5] Dan Sullivan. *IaaS Providers List: Comparison And Guide*. Feb. de 2014. URL: http://www.tomsitpro.com/articles/iaas-providers,1-1560.html.

[6] Wikipedia. *Platform as a Service*. URL: https://en.wikipedia.org/wiki/Platform_as_a_service.

[7] GetCloud Developers. *Benefits And Drawbacks Of PaaS*. Dic. de 2014. URL: http://www.getcloudservices.com/blog/benefits-and-drawbacks-of-paas/.

[8] Dan Sullivan. *PaaS Providers List: Comparison And Guide*. Ene. de 2014. URL: http://www.tomsitpro.com/articles/paas-providers,1-1517.html.

[9] Wikipedia. *Client-Server Model*. URL: https://en.wikipedia.org/wiki/Client%E2%80%93server_model.

[10] *Types of Servers*. URL: https://image.slidesharecdn.com/basicnetworkconcepts-140422021634-phpapp01/95/introduction-to-computer-network-23-638.jpg.

[11] *Request Response Pattern*. URL: http://www.servicedesignpatterns.com/Images/RequestResponse.jpg.

[12] *Model View View Model*. URL: https://www.devexpress.com/Products/NET/Controls/WPF/i/features/mvvm-light.png.

[13] Martin Fowler. *GUI Architectures*. Jul. de 2006. URL: https://martinfowler.com/eaaDev/uiArchs.html.

[14] *The MVVM Pattern*. Feb. de 2012. URL: https://msdn.microsoft.com/en-us/library/hh848246.aspx.

[15] Google. *Angular*. URL: https://angular.io.

[16] Techwall. *What Is Web Protocol?* URL: https://www.techwalla.com/articles/what-is-web-protocol.

[17] Wikipedia. *Hypertext Transfer Protocol*. URL: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol.

[18] *What does the framework mean in Java? And what is difference between framework and Java?* URL: https://www.quora.com/What-does-the-framework-mean-in-Java-And-what-is-difference-between-framework-and-Java.

[19] Wikipedia. *Spring Framework*. URL: https://en.wikipedia.org/wiki/Spring_Framework.

[20] GRPC. *GRPC IO*. URL: http://www.grpc.io/.

[21] Apache. *Apache Maven Project*. URL: https://maven.apache.org/.

[22] *Bulma IO*. URL: http://bulma.io/.

[23] Miguel Arlandy Rodríguez. *Introducción a Apache ActiveMQ*. URL: https://www.adictosaltrabajo.com/tutoriales/active-mq/.

[24] Amazon Web Services. *Amazon SQS*. URL: https://aws.amazon.com/es/sqs/.

[25] Docker. *What Docker?* URL: https://www.docker.com/what-docker.

[26] Amazon Web Services. *Continuous Integration*. URL: https://aws.amazon.com/es/devops/continuous-integration/.

[27] Amazon Web Services. *Amazon Cognito*. URL: https://aws.amazon.com/es/cognito/.

[28] tensorflow. *Inception v3*. URL: https://github.com/tensorflow/models/tree/master/inception.

[29] Maria DB Foundation. *Maria DB*. URL: https://mariadb.org/.

[30] Noticias Jurídicas. *La Ley Orgánica 15/1999, 13 de diciembre, de Protección de Datos de Carácter Personal (LOPD)*. URL: http://noticias.juridicas.com/base_datos/Admin/lo15-1999.html.

[31] AGDP. *Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo*. URL: http://www.agpd.es/portalwebAGPD/revista_prensa/revista_prensa/2016/notas_prensa/news/2016_05_26-ides-idphp.php.

[32] Noticias Jurídicas. *Ley 34/2002, de 11 de julio, de servicios de la sociedad de la información y de comercio electrónico*. URL: http://noticias.juridicas.com/base_datos/Admin/l34-2002.t1.html.

[33] Unión Europea. *Directiva 2000/31/CE del Parlamento Europeo y del Consejo*. URL: http://eur-lex.europa.eu/legal-content/ES/TXT/HTML/?uri=CELEX:32000L0031&from=ES.

[34] Noticias Jurídicas. *Ley 7/1998, de 13 de abril, sobre Condiciones Generales de la Contratación*. URL: http://noticias.juridicas.com/base_datos/Privado/l7-1998.html.